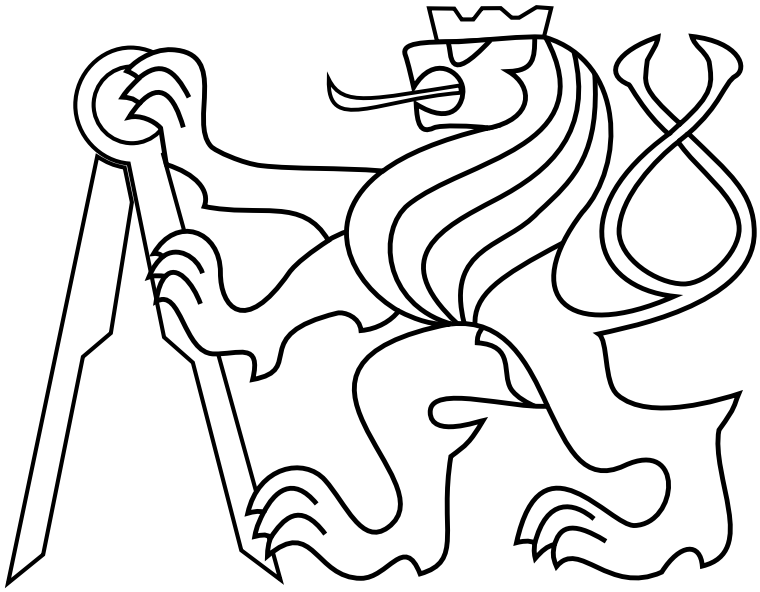
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Matouš Melecký

Monocular 3D Scene Reconstruction for an Autonomous Unmanned Aerial Vehicle

Department of Cybernetics

Thesis supervisor: **Ing. Matěj Petrlík**

May 2021

Author statement for undergraduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date..... .....

Signature

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Melecký Matouš** Personal ID number: **483603**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Monocular 3D Scene Reconstruction for an Autonomous Unmanned Aerial Vehicle

Bachelor’s thesis title in Czech:

Monokulární rekonstrukce 3D scény pro autonomní bezpilotní helikoptéru

Guidelines:

The focus of this thesis is to develop a system that takes images from a monocular camera as input, and from a sequence of such images, reconstructs the 3D structure of the captured scene. The following tasks will be solved:
• Design and implement an algorithm for 3D scene reconstruction [1-2] from a stream of monocular camera images. Pose estimates from a self-localization system [3] of the unmanned aerial vehicle (UAV) can be used to initialize the camera poses.
• Verify the developed scene reconstruction technique on the publicly available EuRoC dataset [4], which contains indoor flight sequences of a UAV equipped with cameras. The provided ground-truth 6DOF pose and structure can be used to evaluate the algorithm performance.
• Integrate the developed method into the Multi-robot Systems Group UAV system [5].
• Conduct experiments in the realistic Gazebo simulator to verify the integrated system.
• Demonstrate the feasibility of deployment onto the actual hardware platform flying in a real-world environment. Discuss results and parameter selection.
• Extend the reconstruction process by a postprocessing step, which can be run offline after the flight to improve the accuracy of the reconstructed structure.
• Evaluate the improvement in the quality of the post-processed structure compared to the structure obtained during the flight.

Bibliography / sources:

[1] Häming, Klaus, and Gabriele Peters. "The structure-from-motion reconstruction pipeline–a survey with focus on short image sequences." Kybernetika 46.5 (2010): 926-937.
[2] Schonberger, Johannes L., and Jan-Michael Frahm. "Structure-from-motion revisited." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.
[3] Geneva, Patrick, et al. "Openvins: A research platform for visual-inertial estimation." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020.
[4] Burri, Michael, et al. "The EuRoC micro aerial vehicle datasets." The International Journal of Robotics Research 35.10 (2016): 1157-1163.
[5] Baca, Tomas, et al. "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles." arXiv preprint arXiv:2008.08050 (2020).

Name and workplace of bachelor's thesis supervisor:

Ing. Matěj Petrlík, Multi-robot Systems, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **08.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Matěj Petrlík	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

Firstly, I would like to thank my great adviser, Ing. Matěj Petrlík, for his kind guidance, plentiful advice, professional motivation and enormous patience. Also, I would like to thank all the members of the MRS group that helped me along the way. Further, I would like to appreciate the excellent education provided to me by the Faculty of Electrical Engineering. Without it, I would never have been able to write this thesis.

I am also very fortunate to have a wonderful caring family, who have supported me all along the way. I am deeply grateful to my extraordinary girlfriend Ivana, who stood by me the whole time, even if I had to work all day and night and little time remained to the two of us.

Lastly, I must also thank our family dog Tim, as walks with him were one of the things that helped me stay sane during the whole process.

Abstract

The real-time 3D reconstruction of the surrounding scene is a key part in the pipeline of the autonomous flight of unmanned aerial vehicle (UAV). The combination of an inertial measurement unit (IMU) and a monocular camera is a common and inexpensive sensor setup that can be used to recover the scale of the environment. This thesis aims to develop an algorithm solving this problem for this particular setup by leveraging the existing visual-inertial navigation system (VINS) odometry algorithms for localisation.
Two algorithms are developed, wide-baseline matching-based and small-baseline tracking-based. Also, an offline post-processing structure-refinement step is implemented to further improve the resulting structure. The algorithms and the refinement step are then evaluated on publicly available datasets. Furthermore, they are tested in a simulator and a real-world experiment is conducted.
The results show that the tracking-based algorithm is significantly more performant. Importantly, tests on the datasets and the real-world experiments suggest that this algorithm can be practically employed in application scenarios.

Keywords: unmanned areal vehicle, 3D scene reconstruction, structure from motion, monocular simultaneous localisation and mapping, non-linear least squares

Abstrakt

Rekonstrukce 3D modelu prostředí je klíčovou částí autonomního letu bezpilotní helikoptéry (UAV). Kombinace inerciální měřicí jednotky (IMU) a kamery je běžnou a dostupnou senzorovou sadou, jež je schopna získat informaci o měřítku prostředí. Tato práce si klade za cíl vyvinout algoritmus řešící problém 3D rekostrukce pro tyto senzory za využití existujících metod vizuálně-inerciální lokalizace (VINS).
V práci jsou navrženy dva algoritmy, odlišené způsobem, jakým extrahují korespondence mezi snímky: párovací algoritmus se širokou bází a algoritmus založený na trackingu s malou bází. Také je implementována metoda vylepšující výslednou 3D strukturu po letu. Algoritmy jsou otestovány na veřejně dostupné datové sadě. Navíc jsou otestovány v simulátoru a je proveden experiment v reálném prostředí. Výsledky ukazují, že algoritmus založený na trackingu dosahuje výrazně lepších výsledků. Navíc testy na datech a experimenty v reálném prostředí ukazují, že algoritmus může být nasazen v praktických aplikačních situacích.

Klíčová slova: bezpilotní helikoptéra, rekonstrukce 3D scény, struktura z pohybu, monokulární lokalizace a mapování, nelineární nejmenší čtverce

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

In this introductory chapter, we provide the motivation of the problem that we will be solving. We follow this by a detailed account of the problem definition, its goals, constraints and other specifics. Then we outline the work's structure.

1.1 Motivation

The UAVs are becoming increasingly popular, both in industry and research. Although UAVs are defined generally as any man-made devices, that can fly without a human operator on board, when we talk about UAVs in this work, we will refer to multi-rotor UAVs.

Recently, the focus in research has been on increasing the autonomy of UAVs as it has enormous potential for many new exciting applications, such as search-and-rescue missions [1], firefighting inside high-rise buildings [2], area monitoring [3], or manoeuvres within close range from objects, which can prove beneficial in industrial settings [4], documentation of heritage [5], and elsewhere.

Even ordinary consumer-grade UAVs have semi-autonomous systems embedded in them, improving their operability and unlocking their potential to a wider audience of operators, which gave rise to their increased popularity in both professional and amateur photography.

Full autonomy of these systems is a complicated feat that requires many problems to be solved. Firstly, UAVs need to be able to properly control their hardware and abstract it. Then, a sense of perception of the world is required, meaning that the UAVs has to be able to localise itself within the world while map it at the same time and then determine positions of points of interest within the built maps. This can then be input to more high-level procedures, such as path planning and decision making. Needless to say, all of this has to happen in real time. It used to be the case that equipment with computation power adequate to the extent of

Figure 1.1: Practical deployment of the autonomous UAV platforms of the Multi-robot Systems (MRS) group. From left to right: testing of autonomous flight in the Býčí skála cave for the DARPA Subterranean Challenge, documenting inaccessible parts of churches for the Czech National Heritage Institute, autonomous flight in the forest environment, extinguishing a fire at the MBZIRC 2020 challenge.

the problem at hand could not be carried by these smaller devices. This is not true anymore, and systems capable of relative autonomy can be of comparably small sizes.

Most of the consumer-grade systems are using Global Positioning System (GPS) for localisation purposes. This solution is not satisfying for many applications, however, because GPS does not offer a fine precision of centimetres and, most importantly, it is not available in many situations at all, such as inside buildings. For this, simultaneous localization and mapping (SLAM) algorithms were developed. Making the autonomous UAVs cheaper, more capable and more reliable are the essential problems needed to solve for widespread adoption of these systems. One of the solutions to this may be the visual SLAM because cameras are fairly cheap and common devices.

Often, the map created by the visual SLAM is not detailed enough for the purposes of autonomous navigation. Therefore, the main aim of this work is to build upon existing SLAM systems, creating a more detailed map of the environment that would be both detailed enough for autonomous planning and also eligible and understandable for human operators overseeing the UAVs.

We will utilise results from both SLAM and also a similar problem called structure from motion (SfM), which is an area of interest for the computer vision community and puts the focus on the quality of the 3D reconstruction.

1.2 Problem definition

Provided with pose estimates from an existing localisation algorithm and images from a monocular camera, we seek to develop an algorithm that is able to reconstruct the map of the environment in real time on a central processing unit (CPU). Crucially, the algorithm must leave enough computational resources for use in the subsequent autonomous path-planning procedures.

More specifically, the algorithm is designed for a UAV equipped with a monocular monocular and an IMU, as this is the minimal setup able to recover the true metric scale of the world, which is important in the robotics' applications.

As UAVs are often used for documentation purposes[1], we would also like to use the resulting map as some lasting output. This would unlock many more possibilities. As the flight conditions are rather restrictive when it comes to computational resources, we want to be able to easily perform a post-processing step after landing that would further refine the 3D structure, using already generated structure and full computational resources.

1.3 Thesis' outline

In this chapter, we have motivated the problem and specified its definition and restrictions.

The following work attempts to solve the problem at hand, covering both its theoretical aspects and practical considerations.

[1] `https://dronument.cz`

First, we introduce the SLAM, SfM and other topics relevant to our problem in chapter 2, seeking to provide the reader with a broad and thorough overview of the state-of-the-art literature.

Then, in chapter 3, we will first state the mathematical notation used throughout the text and lay out important theoretical foundations. We follow this theoretical overview with a description of the proposed algorithms in chapter 4. In chapter 5, we describe the offline post-processing extension to the real-time algorithms.

Lastly, in chapter 6, we evaluate the proposed algorithms. We disscuss the parameter selection in the section 6.2. In section 6.4, we verify the developed scene reconstruction techniques on the publicly available EuRoC dataset [6]. We integrate the methods into the UAV system used by the MRS group [7] and evaluate this integration by performing experiments in the Gazebo simulator in section 6.6. and in a real-world environment in section 6.7. The comparison of the post-processed structure with the structure obtained during the flight is provided in section 6.4.4.

Chapter 2

State of the Art

Contents

In this chapter, we will describe SfM and related problems and present the reader with state-of-the-art methods for solving these problems and other relevant literature. Furthermore, we provide a short overview of depth-completion methods. We seek to provide a broad literature survey to put our problem into context and also to provide possible starting points for future work. As many algorithms rely on graphics processing unit (GPU) for crucial computation, we also provide a discussion of the performance boost that GPU provide and the reasoning why we did not use GPU in our implementation.

2.1 Structure from Motion

The SfM, in general, is the process of recovering 3D structure, usually in the form of a point cloud and camera view poses from a set of images. The set can be ordered, such as images taken from a moving car, or unordered, such as a photo collection on the internet. This has implications on the structure of the algorithm as the order in the set of images can be exploited and taken advantage of. Usually, SfM algorithm is not performed in real time, and its main focus is the quality of the recovered 3D structure. In this work, we would also like to maximise the quality of the 3D structure, but with additional constraint being that the algorithm must be able to perform in real time.

A large body of research has been written about SfM over the years, as it is one of the fundamental problems of computer vision, and it is not possible to list all of the major work. The research on this problem began in 1987 with the paper [8]. Survey [9] compiles more recent advancements in SfM, focusing mainly on pure structure and camera pose recovery. An in-depth summary of the multi-view reconstruction can be found in the book [10] by R. Hartley

and A. Zisserman. Recently, the focus has been on the scalability of the algorithms, with some implementations able to handle millions of photos [11, 12]. On the other hand, in [13] the focus is put on being able to reliably recover the 3D structure even from short sequences of images. The most popular approach to SfM is incremental, which consists of incremental adding of new images into the reconstruction with some examples being [11, 14, 15]. We will outline this approach next, as it also is the only one that can be adapted to a real-time scenario. Other approaches include global [16], and hierarchical methods [17].

More recently, deep-learning approaches to 3D reconstruction have been developed. Some of the recent advancements are described in the survey [18].

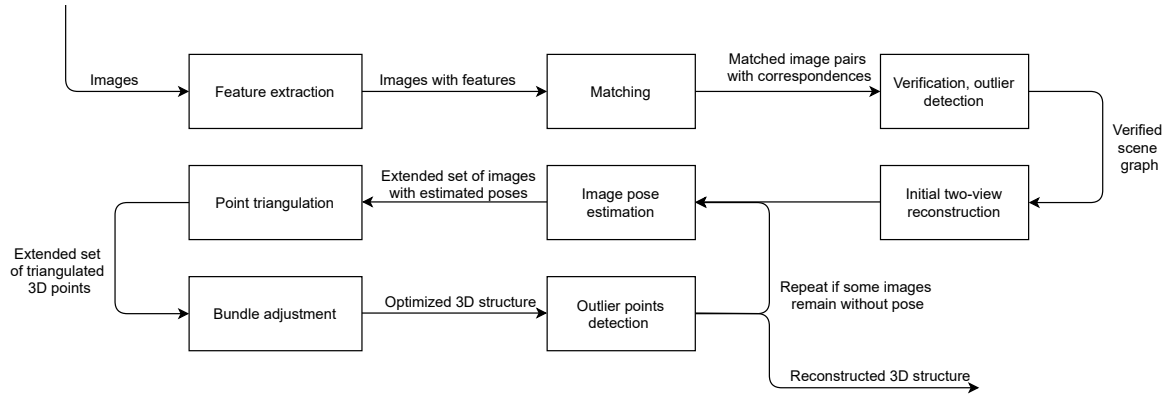


Figure 2.1: Diagram of the incremental SfM pipeline.

The SfM integrates solutions to many individual smaller problems, to produce its result. The incremental SfM pipeline is shown in figure 2.1. First, some feature extractor is run on images, which identifies parts of the image that are in some sense "interesting", e.g. corners, lines. After this, descriptors of each feature are computed. The role of feature descriptors is to accurately describe the area of the image that was identified as interesting so that it can be compared against other areas in other images. Ideally, descriptors should be invariant under geometric and radiometric changes so that the same object could be identified in as many images as possible. They should also be discriminative, however, to prevent false matches.

The scale-invariant feature transform (SIFT) [19] is one of the most popular algorithms for this task. It can handle both the feature extraction and also the computation of descriptors and is robust. On the other hand, it is resource-expensive and slow [20] which bars its usage in real-time applications. Algorithms with binary features, such as Oriented FAST and Rotated BRIEF (ORB) [21], offer better performance in terms of speed while keeping the property of rotational invariance. The recent trend in this area has been to use learning techniques for defining or improving feature descriptors and detectors such as SuperPoint [22].

The next step is to match the descriptors against other images. This step is where an ordered photo collection can be helpful, as it often can be reasonable to assume, that images far from each other in the ordering are also far away from one another in reality. Otherwise, we must match either naively every image to every other image, or use some algorithms that can improve the efficiency of this process [23].

This constructs the so-called scene graph. Its nodes are the images, and edges represent the match between two images and their descriptors. The purpose of the scene graph building

process is to identify images that might view the same real object. Various strategies can be applied to filter out outliers in matches, most notably random sample consensus (RANSAC) and its derivatives [24, 25]. Matches can also be geometrically verified by estimating a transformation that maps features between the two images. This mapping can either be described by epipolar geometry or by a homography in the case of co-planar points [10]. A pair of images is considered to be geometrically verified if there exists such a valid transformation that maps enough correspondences between them.

Then, a pair of images is chosen to initiate the reconstruction and set the scale by a two-view reconstruction. It is important to choose a high-quality initial pair, as the initialisation has a large impact on the rest of the reconstruction. In real-time applications, however, we do not have much control over what the first pair will be. In regular SfM, the absolute metric scale cannot be recovered and is fixed by arbitrary initialisation. To fix the scale and remove the ambiguity, we would need some additional information, which in this work we will have, see section 2.2.2.

When the initial two-view reconstruction is performed, new images and scene points are added by solving the Perspective-n-Point (PnP) problem [26], that uses correspondences of already triangulated 3D points to points in the newly added image to estimate the camera pose of this image. Meanwhile, triangulation is performed to reconstruct more points. A point can be triangulated if it is observed by at least two already registered images. Triangulation can be only two-view or more general and precise m-view. Many algorithms for multi-view triangulation exist, and some of them are listed in the book [10]. In this work, we describe one triangulation algorithm in section 4.1.5.

The two last steps provide us with a rough estimate of the camera poses and 3D points. These estimates are then refined by using a non-linear optimisation method called the bundle adjustment (BA). It is trying to find such camera parameters and point positions that minimise the reprojection error. This is, in essence, a sparse geometric parameter estimation problem [27]. Usually, the BA is modelled as a non-linear least square problem [14], although this is not a general assumption, and other types of models might be suitable as well [27].

Some of the parts of the SfM will be explained in greater detail further on in the work. As we are trying to solve SfM in real time, some parts of the algorithm will be handled by different procedures. Most importantly, we will not be directly solving the PnP problem in this work, and delegate most of the camera pose estimation work to other procedures that will be described in the following section 2.2. Our focus will be mostly on correspondence matching, triangulation and BA.

2.2 SLAM and Visual-inertial Estimators

SLAM is an important problem in robotics and perception, where a robot has to localise its position within the environment whilst also building a map of it [28]. This has to be done in real time, which imposes strict constraints on the possible solutions. The applications of solving SLAM go well beyond the robotics community; one example might be augmented reality [29]. One advantage of a device using SLAM is that it can work under many different settings, where localisation based on other external sources like GPS would not work. This especially holds for indoor or enclosed areas or generally for areas with poor signal reception.

In some regards, the SLAM is somewhat similar to SfM. It, too, recovers poses of camera views and some structure of the environment. However, SLAM is focused more on the recovery of camera poses and hence the robot's path, rather than the detailed 3D structure. Also, it is an online problem when SfM, usually, is not.

Generally, a SLAM system works by integrating information from odometry (e.g. wheel rotation) and some perception sensors, recovering the depth information and creating 3D landmarks that can be used as reference points for recovering the pose in time. SLAM frameworks tend to be probabilistic, as they have to deal with large uncertainty which arises from real-time conditions, sensor inaccuracies and noise, and localisation errors. Some statistical framework is used to propagate the uncertainty, notably extended Kalman filter (EKF) [28].

Many variants of SLAM exist, usually stemming from some configuration of available sensors. The first sensors, for which SLAM was designed, were active sensors, such as light detection and ranging (LIDAR) [30], which have the advantage of fine precision when recovering depth information and also a 360° field of view. This comes at a price, however, as they tend to be expensive, large and also resource-intensive, which often prohibits their use in more constrained real-world conditions.

2.2.1 Visual SLAM

A major alternative to LIDARs are regular cameras [31]. Especially, stereo-camera setups with a static baseline can be used for depth estimation. A great advantage to using camera-only SLAM (visual SLAM) is that camera is a widely used piece of equipment that tends to be already available in many scenarios as they usually can serve a multitude of different purposes. Processing of the camera output is also well studied by the computer vision community.

Disadvantages of the visual SLAM lay in a limited field of view and more complicated data structure, making them a harder problem than SLAM with LIDAR [31].

Feature-based visual SLAM methods

The more popular and traditional type of visual SLAM systems is the feature-based approach, which is similar in essence to that of the SfM. These methods reduce the image to a set of features (usually keypoints) and then perform their matching or tracking over multiple frames. This is indeed a similar approach to the traditional SfM and is the most often used solution.

The first of these algorithms was PTAM [32]. It was also the first to introduce the architecture of two threads, one for tracking and the other for optimisation, that we also use in this work. The approach is focused on the applications in small-scale augmented reality (AR), which means that it does not focus on the density of the maps and can only track and map over a small space.

One of the modern examples of this approach is ORB-SLAM [33] that is based, as its name suggests, on ORB descriptors. Our work is largely based on these approaches, as they achieve real-time performance and have been proven to be robust to changes in illumination and rapid movements. By decoupling the mapping from the tracking, we seek to achieve a denser reconstruction of the environment.

Direct and dense SLAM methods

The direct approaches to the visual SLAM problem do not compute keypoints, but rather base their solution on finding the alignment between two frames by minimisation of the photometric error over the whole image [34]. The minimisation over the whole image, such as in [35], achieves impressive results in terms of the density of the reconstruction; however, it cannot be run in real-time on a CPU but rather requires state of the art GPU, which is problematic on board of the UAV. Another limitation of this approach is that it is not able to perform at a frame-rate required in our problem and cannot handle large motions well.

Versions of this approach exist that are capable of running in real time on a CPU, such as LSD SLAM [36] or DSO [37]; however, the density has to be sacrificed. These methods focus only on parts of the image with sufficient texture information, as those are the easiest to optimise. Still, these methods suffer from the loss of tracks in large movements. However, we believe that these methods are a viable alternative to the feature-based approaches, and if the GPU technology advances, we may witness an advent of these methods in more challenging environments. Especially the achieved density has a potential to significantly improve state of the art.

In one of our methods, we used the idea from [38] of approximating the local relationships between points by a Delaunay triangulation, which is also based on the direct approaches. They reformulate the photometric error optimisation to a graph optimisation problem over the Delaunay triangulation.

2.2.2 VINS

In this work, we are interested in monocular systems (having a single camera) coupled with an IMU. This is a sensor that, using a combination of gyroscopes and accelerometers, measures angular rate, specific force and orientation. For the combined cameras and IMU setups, a class of algorithms called visual-inertial navigation system (VINS) has been developed [39, 40, 41]. In the following, we will focus on the monocular VINS case.

The main benefit over the regular monocular-camera SLAM is the increased accuracy and observability of not only the pitch and roll angles, but the metric scale as well [40], meaning that the resulting reconstruction is in the scale of the world that it captures, which is essential for robotics. IMU measurements are also very useful in situations such as loss of visual track due to poor visibility, motion blur, or area with not enough texture, where vision-only approaches might otherwise fail [40]. Also, it suffices to use a small and low-cost IMU. However, there are some limitations to this technology, too. In order for the metric scale to be observable, acceleration is needed. Therefore monocular VINS estimators need to launch from some moving state rather than stationary. This means that after the start of a device, there is some short window where localisation cannot be performed. On the contrary, stereo-camera systems have no issues with starting from stationary positions. Also, correct camera-IMU extrinsic calibration has to be carried out prior to using the system [40].

As this framework is focused mainly on localisation, and maps produced by it are too sparse for many applications, the aim of this work is to build upon its results to produce a more detailed map that would be more adequate in other tasks, such as drone navigation. We will leverage the timestamped position data produced by VINS estimators to build this

map in real time. In the following, we will briefly introduce the VINS algorithms chosen for the evaluation of the algorithms and provide a reason as to why they were chosen.

OpenVINS

OpenVINS [41] is an open-source implementation of a VINS algorithm based on an on-manifold EKF that will be used for the evaluation of the methods presented here. We chose this implementation because of its extensive documentation that greatly simplifies the extension process. It has also achieved convincing results on the EuRoC dataset [6]. Also, this dataset can be taken advantage of for our implementation, too, as it also provides the ground truth 3D structure measurements performed by a precise 3D scanner.

Another advantage of OpenVins is that it supports a wide range of hardware configurations, such as multiple IMUs, cameras and different feature representations.

VINS-mono

VINS-mono [40] is the system currently employed in the MRS group. It is focused on the monocular VINS problem. In contrast to the OpenVINS it uses a non-linear optimisation of the pose-graph instead of an EKF filter, supports loop closure and is based on keyframes.

2.3 Depth completion

During the implementation of the algorithm, we encountered the problem of densification of the produced structure. We were motivated by the need to be able to discern the empty space from obstacles. When the point cloud was too sparse, we did not have enough information. This problem brought to our attention the topic of depth completion.

Most of the recent methods solving this problem are based on deep neural networks. These include methods [42] and [43] that use both the RGB image of the scene and the sparse depth map as input to the deep neural network that produces the full depth map. Importantly, they are suited to sparse depth output of the SfM and SLAM algorithms; hence they can also be applied in our case. In [44], the dense depth output is also utilised to improve the result of a direct SLAM method. Unfortunately, the trained models of these methods are fairly large and require the use of GPU for inference.

Depth inference from a single monocular image is another recently studied problem that can be of interest in our case. In this area, more lightweight models that allow real-time inference on a CPU have been developed, usually based on the Mobilenet network. Comparison of various lightweight methods can be found in [45]. We have tested the approach of model FastDepth [46], and we found that it is able to handle the real-time conditions well if the resolution of the image is slightly decreased. However, we found that the results were sometimes too arbitrary and unstable, as the network only observes static images. Even if a depth-map estimate in one frame was relatively accurate, the estimate in the next frame could be arbitrarily different. We believe that this can be a general problem with the neural-based approaches, although admittedly, in the case where sparse depth is leveraged, it is not as

profound. Furthermore, the performance of the networks is largely dependent on the datasets they were trained on, and they may fail entirely in new environments.

Approaches using classical optimisation methods exist, even though it is not clear if they provide major advantages over performing the optimisation over multiple frames as is done in some of the dense direct SLAM methods. Usually, if we require to use these methods on the CPU, they only utilise the sparse depth. The method [47] can run on the CPU in real time; however, it is designed to fill point clouds produced by LIDAR sensors that tend to be more dense and regular. Method [48] is based on the research from the field of compressive sensing and uses L1 minimisation scheme. However, it turned out to be too slow for our use case.

We also attempted to design a method based on image segmentation and following interpolation of similar segments. However, we were not able to reach the desired speed and results. In the end, we used the idea from [38] of using the Delaunay graphs and devised a simple Delaunay-based interpolation scheme that we describe in section 4.2.3. We use it mainly to densify the point cloud, but it can be used for full depth map estimation as well.

2.4 Use of GPU

During our literature research, we came across many methods that have the potential of significantly improving the performance of the SfM algorithms and the density of the produced structure but that rely on the use of state-of-the-art GPUs. The application area of a GPU within the SfM problem is very broad, ranging from neural-powered approaches for many geometry-inference tasks, such as depth completion, and optimisation tasks in the direct approaches to more basic image processing tasks, which often are even embarrassingly parallel. With the advent of new lightweight GPUs, such as Nvidia Jetson TX2, these approaches become more feasible in smaller devices. However, at the moment, these are still expensive and consume relatively large amounts of power. Even as they become progressively smaller, we believe that algorithms that can run with only a regular CPU will still have their place, for example, in more miniature robots.

Chapter 3

Theoretical Foundations

Contents

In this chapter, we introduce the fundamental concepts of the projective camera geometry, stereo camera relationships and non-linear least-squares optimisation. We deem these topics essential for the SfM algorithm, and they appear in multiple parts of the solution. Reader with prior knowledge of these topics is encouraged to skip to chapter 4 describing the algorithm itself. Also, for the comfort of the reader, we provide a short reference of the mathematical notation we are using.

3.1 Mathematical notation

In the table 3.1, we outline the mathematical notation used throughout this work. Please also note that we use terms vector and point interchangeably.

Symbol	Meaning
x	scalar
$\mathbf{x}$	column vector in Cartesian coordinates
$\bar{\mathbf{x}}$	vector in homogeneous coordinates
$h : \mathbb{R}^{n+1} \to \mathbb{R}^n$	conversion from homogeneous coordinates defined as: $h([\mathbf{x}\|l]^T) = \mathbf{x}/l$
$\|\|\mathbf{x}\|\|$	the Euclidean norm of the vector $\mathbf{x}$
$f(\mathbf{x}; \mathbf{p})$	function of variable $\mathbf{x}$ parametrized by the parameters $\mathbf{p}$
$\mathbf{X}$	matrix
$\mathbf{X}^T$	transpose
$\mathbf{X}^{-1}$	matrix inverse
$\det(\mathbf{X})$	determinant of the matrix $\mathbf{X}$
$\mathbf{I}$	identity matrix, ones on the main diagonal, otherwise zeroes
$\mathbf{0}$	zero vector or matrix, depending on the context
$\mathbb{R}$	set of real numbers
$\mathbb{R}^{m \times n}$	matrices of real numbers with dimensions $m \times n$
x_t	some information x in time t
${}^i\mathbf{x}$	vector in some coordinate frame i
$\mathbf{x}(i)$	value at the i'th coordinate of vector $\mathbf{x}$
$\mathbf{R}_{i \to j}$	the rotation component of a transform from frame i to j
$[\mathbf{a}]_\times$	the matrix representation of a vector product: $\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b}$, see (3.10)
Ω	the set of pixel coordinates of an image, $\Omega \subset \mathbb{Z}^+ \times \mathbb{Z}^+$
$I : \Omega \to \mathbb{R}$	the image grayscale intensity value function

Table 3.1: Mathematical notation

3.2 Camera

A camera can be mathematically described as some mapping from 3D coordinates to a 2D image. This mapping is specified by a camera model, which has some form and parameters, that approximate the actual camera being described. In the section 3.2.1, we describe the commonly used basic pinhole model. The pinhole model does not account for many common distortions of the lenses. Therefore we introduce distortion parameters in section 3.2.2.

In section 3.2.3, we define the concept of the depth of a point.

3.2.1 Pinhole camera model

In the pinhole camera model, we consider a central projection on the image plane with the projection centre $\mathbf{c} \in \mathbb{R}^3$ called the camera centre. Suppose we have a point $\mathbf{x} \in \mathbb{R}^3$, expressed in the camera coordinate frame, meaning that $\mathbf{c}$ is the origin of the Cartesian system of coordinates and image plane is defined as $z = f$, where f is the focal length of the

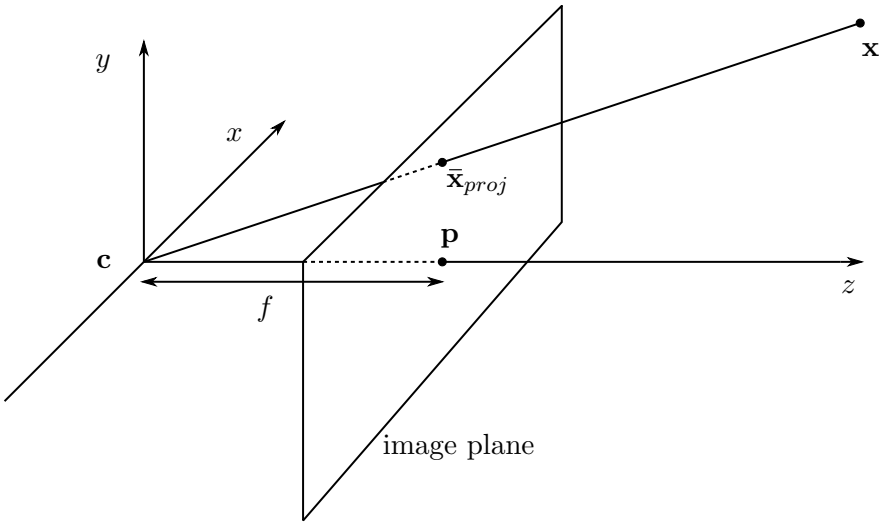


Figure 3.1: Pinhole camera model.

camera. Consider the homogeneous representation $\bar{\mathbf{x}}$ of the point $\mathbf{x}$. The central projection onto the image plane in homogeneous coordinates $\bar{\mathbf{x}}_{proj} \in \mathbb{R}^3$ is then defined as

$$\bar{\mathbf{x}}_{proj} = \mathbf{K}[\mathbf{I}|\mathbf{0}]\bar{\mathbf{x}}, \tag{3.1}$$

where matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$ is called the calibration matrix and is defined as

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.2}$$

(p_x, p_y) are the coordinates of the principal point $\mathbf{p}$ in the image plane. The situation is captured in the figure 3.1. Camera with $f = 1$ and $\mathbf{p} = [0, 0]$ is called the normalised camera.

We will deal with problems, where we will need multiple cameras, therefore it is more practical to keep points in some global coordinate frame. For camera centre ${}^g\mathbf{c}$ and point ${}^g\bar{\mathbf{x}}$, both in the global coordinate frame, the equation (3.1) then becomes

$$\bar{\mathbf{x}}_{proj} = \mathbf{K}\mathbf{R}[\mathbf{I}| - {}^g\mathbf{c}]{}^g\bar{\mathbf{x}}, \tag{3.3}$$

for some rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ representing the orientation of the camera. Parameters found in matrix $\mathbf{K}$ are called intrinsic parameters, while $\mathbf{R}$ and ${}^g\mathbf{c}$ are called extrinsic parameters. The expression that ${}^g\bar{\mathbf{x}}$ is multiplied by can be interpreted as a matrix $\mathbf{P} \in \mathbb{R}^{3\times4}$. We call $\mathbf{P}$ the full camera matrix. If we leave out $\mathbf{K}$ out of the matrix computation, which is equivalent to using a normalised camera, we call the resulting matrix the extrinsic matrix.

A more compact representation can be achieved if we define a vector $\mathbf{t} = -\mathbf{R}{}^g\mathbf{c}$. We can then write (3.3) as

$$\bar{\mathbf{x}}_{proj} = \mathbf{K}[\mathbf{R}|\mathbf{t}]{}^g\bar{\mathbf{x}}. \tag{3.4}$$

The model of the matrix $\mathbf{K}$ above assumes that the image coordinates are in the euclidean system with equal scales in directions of both axes. However, in reality, we often

measure the image coordinates in pixels, which is referred to as pixel coordinates. Also, the number of pixels in both directions per a unit distance in image coordinates can be different. Let us denote this by m_x in the direction of x-axis and m_y in the direction of y-axis. Then for the pixel coordinates the matrix $\mathbf{K}$ becomes:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.5}$$

where $f_x = fm_x$ and $f_y = fm_y$ represent the focal length in the pixel dimensions of both axes. The focal point is expressed in pixel coordinates as $c_x = p_x m_x$ and $c_y = p_y m_y$.

A more detailed description of the pinhole camera model can be found in [10].

3.2.2 Distortion

Usually, the pinhole camera model is not precise enough. Namely, it does not account for the distortion effects of the lens. Therefore we also use a distortion model, with the most common being radial distortion. In general, distortion occurs when camera projections of straight lines are no longer completely straight but get some curvature. The example of this can be seen in figure 3.2.

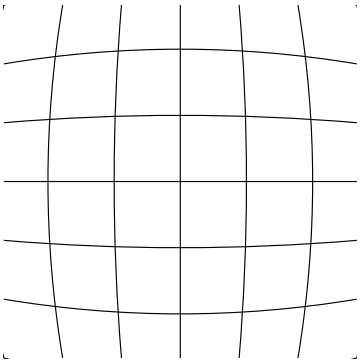


Figure 3.2: Illustration of lines under distortion.

Radial distortion model uses the fact, that often distortions are approximately radially symmetric. When we measure a point $\mathbf{x} \in \mathbb{R}^2$ in pixel coordinates, we can write the correction as

$$\hat{\mathbf{x}} = \mathbf{x}_c + L(r)(\mathbf{x} - \mathbf{x}_c), \tag{3.6}$$

where $\mathbf{x}_c$ is the centre of radial distortion which usually is the principal point and $L(r)$ is the distortion factor, which is a function of the radial distance $r = \|\mathbf{x} - \mathbf{x}_c\|$.

The function $L(r)$ is given as a Taylor expansion $L(r) = 1 + \kappa_1 r + \kappa_2 r^2 + \kappa_3 r^3 + \ldots$ and its coefficients $\{\kappa_1, \kappa_2, \kappa_3, \ldots\}$ are then the radial distortion parameters. They are retrieved in the calibration process and there are multiple methods available [10].

In practice, we often want to model both radial and hyperbolic distortion, which can be modelled by the Brown–Conrady model [49].

Fisheye distortion

Fisheye distortion model describes the fisheye cameras. These cameras are of interest in the UAV scenario as they have a wide field of view. This is especially useful for quick manoeuvres as the UAV tends to tilt downwards during the forward acceleration, and the regular camera would not be able to see what is in the front of the vehicle.

Let point $\hat{\mathbf{x}}$ be a projection of some point in image coordinates using the pinhole camera model. Next we apply the fisheye distortion to get the distorted projection $\mathbf{x}$:

$$\mathbf{x} = \frac{\theta_d}{||\hat{\mathbf{x}}||}\hat{\mathbf{x}}, \tag{3.7}$$

where θ_d is defined as

$$\theta_d = \theta(1 + \kappa_1\theta^2 + \kappa_2\theta^4 + \kappa_3\theta^6 + \kappa_4\theta^8). \tag{3.8}$$

$\kappa_1, \kappa_2, \kappa_3, \kappa_4$ are the fisheye distortion coefficients and $\theta = \arctan(||\hat{\mathbf{x}}||)$. The details of this model are described in [50].

3.2.3 Depth of a point

Consider a point $\mathbf{x} \in \mathbb{R}^3$ and an extrinsic matrix $\mathbf{P} \in \mathbb{R}^{3\times4}$. The depth d of a point is defined as the last coordinate of the vector $\mathbf{P}\bar{\mathbf{x}}$.

3.3 Parametrizations of rotations

The most used parametrization of rotations is a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$. The rotation matrices have some specific properties; namely, their determinant is equal to one, and they are orthogonal. All such matrices form the Special Orthogonal Group $SO(3)$.

The representation of rotations as matrices is not the most compact representation, as rotations have 3 degrees of freedom. Because of this and the aforementioned restrictions on the matrices that are hard to enforce in optimisation algorithms we will need to solve, it is useful to use some other representations.

There are multiple other alternatives possible, most notably Euler angles and unit quaternions, but here we focus on the vector representation derived from the Euler–Rodrigues formula.

The representation is given as a rotation by angle θ radians around the axis $\mathbf{a} \in \mathbb{R}^3$, $||\mathbf{a}|| = 1$. The rotation vector is then defined as: $\mathbf{r} = \theta\mathbf{a}$. Furthermore, the matrix representation $\mathbf{R}$ of the rotation $\mathbf{r}$ can be obtained by the Euler–Rodrigues formula:

$$\mathbf{R} = \cos\theta\mathbf{I} + \sin\theta[\mathbf{a}]_\times + (1 - \cos\theta)\mathbf{a}\mathbf{a}^T, \tag{3.9}$$

where the matrix representation $[\mathbf{a}]_\times$ of the vector product with $\mathbf{a} = [a_1, a_2, a_3]^T$ is defined as:

$$[\mathbf{a}]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{3.10}$$

The inverse process requires the choice of the axis **a** from the two alternatives. This can, however, be accomplished by a simple procedure, for which we refer the reader to [51]. For the derivation of the formula, please see [52].

3.4 Epipolar geometry

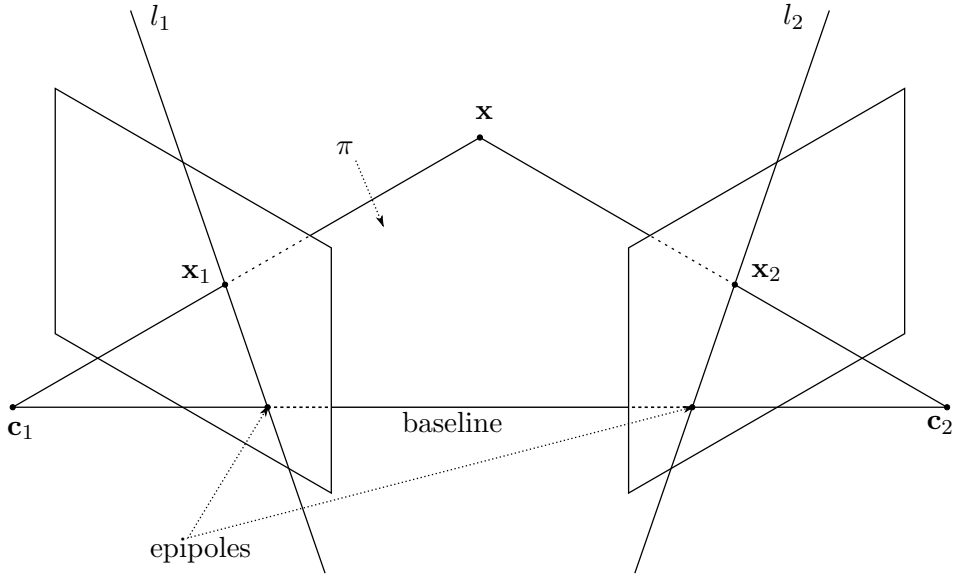


Figure 3.3: Epipolar geometry.

A crucial aspect of the SfM algorithms is the concept of geometric relations between two camera views. These relations are called the epipolar geometry and have practical application in the correspondence search, as they can constrain the solutions.

Suppose we have a point $\mathbf{x} \in \mathbb{R}^3$ that is seen in two views as image points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^2$. It is clear from figure 3.3 that the point $\mathbf{x}$, its projections and camera centres $\mathbf{c}_1, \mathbf{c}_2$ lie in the same plane π called the epipolar plane. The baseline is a line connecting the two camera centres. The point of intersection of the baseline and the image plane is called the epipole. Importantly, epipolar line is the intersection of the epipolar plane with the image plane. Traditionally, the two views are described as a left view and a right view, which stems from the fact that epipolar geometry is often used in stereo vision.

If we only know the poses of the cameras in 3D space and the coordinates of the image point $\mathbf{x}_1$, we can determine the epipolar plane π using these points. Then the corresponding point $\mathbf{x}_2$ must lie on the epipolar line l_2. This fact can be utilised to restrict the correspondence search to the epipolar line or, more practically, to points within some distance from the line. It can also geometrically verify matches computed based on some descriptors. In the following paragraphs, we will derive the algebraic form of this constraint, called the essential matrix.

Essential matrix The camera coordinate systems of the two views can be related by a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$. This means that the relationship between the coordinates of $\mathbf{x}$ in these systems can be written as:

$$^{c_2}\mathbf{x} = \mathbf{R}^{c_1}\mathbf{x} + \mathbf{t}. \tag{3.11}$$

The translation $\mathbf{t}$ corresponds to the directional vector of the baseline. We can demand the following condition to hold:

$$^{c_2}\mathbf{x}(\mathbf{t} \times \mathbf{R}^{c_1}\mathbf{x}) = 0. \tag{3.12}$$

This condition expresses the co-planarity of the baseline and the two lines connecting the point to both camera centres. Because the homogeneous projections $\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2 \in \mathbb{R}^3$ of $\mathbf{x}$ correspond to the same normalised vectors as $^{c_1}\mathbf{x}$ and $^{c_2}\mathbf{x}$, we can rewrite this condition to:

$$\bar{\mathbf{x}}_2(\mathbf{t} \times \mathbf{R}\bar{\mathbf{x}}_1) = 0, \tag{3.13}$$

or, equivalently:

$$\bar{\mathbf{x}}_2^T\mathbf{E}\bar{\mathbf{x}}_1 = 0, \tag{3.14}$$

where

$$\mathbf{E} = [\mathbf{t}]_\times\mathbf{R} \tag{3.15}$$

is the essential matrix. Equation (3.14) is the normal equation of the epipolar line we were seeking.

The essential matrix relates the undistorted points in homogeneous normalised coordinates. We can generalise it to points $\mathbf{p}_1, \mathbf{p}_2 \in \Omega$ in pixel coordinates by utilizing the camera matrices $\mathbf{K}_1, \mathbf{K}_2$:

$$\mathbf{p}_2^T\mathbf{K}_2^{-T}\mathbf{E}\mathbf{K}_1^{-1}\mathbf{p}_1 = 0. \tag{3.16}$$

The matrix

$$\mathbf{F} = \mathbf{K}_2^{-T}\mathbf{E}\mathbf{K}_1^{-1} \tag{3.17}$$

is then called the fundamental matrix.

The essential and fundamental matrices can also be estimated from correspondences, usually by using some robust estimation method such as RANSAC to deal with outliers.

3.5 Non-linear least squares

In the solution of the SfM problem, we often encounter the need to solve a non-linear least squares problem. Therefore, we present the reader with a short overview of the essential theory from this area. The non-linear squares optimisation is a problem of finding $\mathbf{x} \in \mathbb{R}^n$ that minimizes the following functional:

$$\frac{1}{2}\sum_{i=1}^{m} ||f_i(\mathbf{x})||^2, \tag{3.18}$$

where f_i are some general non-linear functions. We note that the constant $\frac{1}{2}$ is not a crucial part of the problem, it is only included as it simplifies the derivatives and it has no effect on the result of the optimisation.

Let us also introduce a matrix notation of the optimisation problem:

$$\underset{\mathbf{x}}{\operatorname{argmin}}\frac{1}{2}||F(\mathbf{x})||^2, \tag{3.19}$$

where $F : \mathbb{R}^n \to \mathbb{R}^m$ is defined as $F(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})]^T$. It can be easily seen that this formulation is equivalent to the initial formulation (3.18).

It can also be useful to constrain the optimisation by setting lower and upper values of $\mathbf{x}$:

$$\begin{aligned} \underset{\mathbf{x}}{\text{argmin}} \quad & \frac{1}{2}||F(\mathbf{x})||^2, \\ \text{s. t.} \quad & \mathbf{d} \leq \mathbf{x} \leq \mathbf{u}. \end{aligned} \tag{3.20}$$

3.5.1 Solving non-linear least squares

The non-linear least squares do not have a closed-form solution, but many iterative methods exist. Here we will describe the trust-region minimisation methods. In the following description, we use the matrix problem notation (3.19).

In general, the iterative methods are based on linearising the problem and computing at each step a correction $\Delta\mathbf{x}$ to $\mathbf{x}$. The linearisation is:

$$F(\mathbf{x} + \Delta\mathbf{x}) \approx F(\mathbf{x}) + J(\mathbf{x})\Delta\mathbf{x}, \tag{3.21}$$

where $J(\mathbf{x})$ is the Jacobian of F. Then, for known $\mathbf{x}$, we need to solve the problem:

$$\underset{\Delta\mathbf{x}}{\text{argmin}} \frac{1}{2}||F(\mathbf{x}) + J(\mathbf{x})\Delta\mathbf{x}||^2. \tag{3.22}$$

This is a linear least-squares problem that can be solved either by exact methods such as QR decomposition, or by some other iterative algorithm [53].

The whole problem (3.19) can be solved by simply iteratively updating $\mathbf{x} := \mathbf{x} + \Delta\mathbf{x}$. However, this solution can have problems with convergence, which is the motivation behind the trust-region methods. Trust region methods restrict $\Delta\mathbf{x}$ to some local neighbourhood, defined by a local distance matrix $\mathbf{D}(\mathbf{x})$ and radius μ, which is dynamically manipulated during algorithm's iterations. The neighbourhood is captured by adding a following condition:

$$||\mathbf{D}(\mathbf{x})\Delta\mathbf{x}||^2 \leq \mu. \tag{3.23}$$

The whole trust region optimisation is then captured in algorithm 1. The value of q measures how well the linearisation corresponds to the original function. If the approximation is poor, the solution is not accepted, and the region radius μ is decreased. On the other hand, if it is very precise, we enlarge the radius. This is controlled by parameters ϵ, η_1, η_2.

The way the constrained linear-squares problem is computed gives rise to different trust-region methods, with the most well-known being the Levenberg-Marquardt. For a more detailed treatment of these methods, we refer the reader to [53]. In most of the non-linear least-squares problems we need to solve in this work, we leverage the Ceres solver [54].

Algorithm 1: Trust region optimisation

Given initial $\mathbf{x}$, radius μ and constants ϵ, η_1, η_2;
while *Termination condition not satisfied* **do**
  Compute $\Delta\mathbf{x}$ by solving the linear least-squares problem (3.22) constrained by (3.23);
  $q = \frac{||F((x)+\Delta\mathbf{x}||^2 - ||F(\mathbf{x})||^2)}{||F(\mathbf{x})+J(\mathbf{x})\Delta\mathbf{x}||^2 - ||F(\mathbf{x})||^2}$;
  if $q > \epsilon$ **then**
    $\mathbf{x} = \mathbf{x} + \Delta\mathbf{x}$;
  end
  if $q > \eta_1$ **then**
    $\mu = 2\mu$;
  end
  else if $q < \eta_2$ **then**
    $\mu = \mu/2$;
  end
end

3.5.2 Robustification

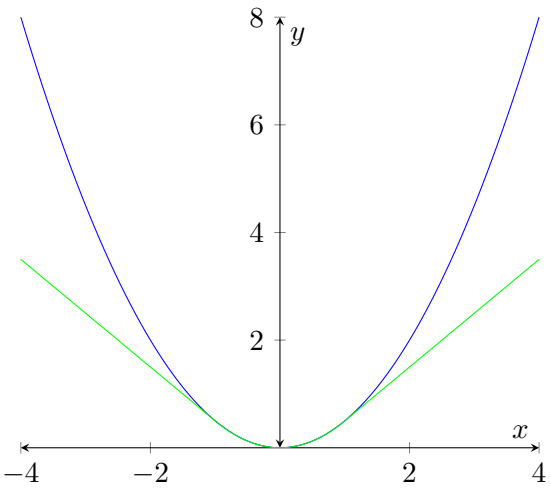


Figure 3.4: Comparison of the quadratic loss $\frac{x^2}{2}$ (blue) and the Huber loss $\rho(x; 1)$ (green).

In real problems, we must deal with outliers, which are erroneous data points with an impact on the value of the loss function, that can steer the optimisation away from the desired solution. We can improve the robustness by replacing the quadratic loss function in (3.18) with a more sophisticated loss function $\rho : \mathbb{R} \to \mathbb{R}^+$ that down-weighs the impact of outliers:

$$\sum_{i=1}^{m} \rho(||f(\mathbf{x})||) \tag{3.24}$$

We want ρ to behave similarly to the original quadratic loss around the zero point, as it is differentiable, which improves convergence, but at the same time decrease the slope further

from the 0. We use the Huber loss function:

$$\rho(a;\delta) = \begin{cases} \frac{1}{2}a^2, & \text{if } \sqrt{a} \leq \delta \\ \delta(|a| - \frac{\delta^2}{2}) & \text{otherwise} \end{cases}. \tag{3.25}$$

See figure 3.4 for a visual comparison of the quadratic and Huber loss functions.

The use of a robust loss function ρ naturally has implications on the solution of the problem, where it affects the computation of the gradient and Hessian. This is out of the scope of this work, and we refer the reader to [27] for more details.

Chapter 4

Proposed Algorithms

Contents

In this chapter, we will outline the structure and theory of the proposed algorithms. The algorithms are designed as an extension to some monocular VINS algorithms, or, more generally, any localisation algorithm or system. It is therefore assumed that the camera pose in time is provided. We then use this information to compute the 3D structure.

The camera is assumed to be calibrated, meaning that the calibration matrix $\mathbf{K}$ does not change in time (e.g. by zooming) and is specified as a parameter at the beginning.

We always begin with a section describing the structure of the whole algorithm, and then we attend to individual details.

4.1 Wide-baseline keyframe-based matching algorithm

This algorithm works on a keyframe basis, which means that we do not consider each image coming from the camera but rather subsample this stream. This has two main reasons. Firstly, we can afford to spend more computation time on each keyframe, hence achieving better results in feature extraction and mapping. Secondly, if we took two images that would be too close to each other, we would have a small baseline, which would cause poor results in the structure estimation.

4.1.1 Algorithm's structure

Each keyframe at time t can be described as a tuple

$$k_t = (\mathbf{C}_t, \mathbf{d}_t, \mathbf{f}_t), \tag{4.1}$$

where matrix $\mathbf{C}_t \in \mathbb{R}^{3\times4}$ is the camera extrinsic matrix that describes camera pose at that keyframe, see eqquation (3.4). Then two vectors, $\mathbf{d}_t$ and $\mathbf{f}_t$ represent computed feature descriptors and the corresponding feature pixel coordinates, respectively.

The state of the algorithm can be described by a vector of keyframes $\mathbf{k}$ and a vector of correspondences descriptions which keep the information about matched images, effectively representing the scene graph. Each correspondence descriptor has all the information needed to perform triangulation of a point. It is defined as follows:

$$d = (\mathbf{key}, \mathbf{desc}), \tag{4.2}$$

where **key** is a vector of indexes of keyframes k_t that have been found to match and **desc** is a vector of indexes of the actual matches $\mathbf{d}_t, \mathbf{f}_t$ for the given k_t.

Assume a given timestamp t. The algorithm receives information about the camera pose at that time, which is represented by a rotation transform $\mathbf{R}_t$ and a position of the camera centre $\mathbf{c}_t$. We then decide whether to add another keyframe. We do this simply by thresholding the translational and rotational distance between the two frames and we describe this in section 4.1.2.

Suppose we selected the timestamp t. We can then initialise a new keyframe. From a corresponding image we compute its features and descriptors $\mathbf{f}_t, \mathbf{d}_t$, which we discuss in detail in section 4.1.3. For feature extraction, we chose the ORB algorithm. SIFT has proven to be too slow for the purposes of a real-time deployment [20].

Then we perform the feature matching as we describe in section 4.1.4. If the number of keyframes is relatively low, we can match with n nearest keyframes, as the search for nearest keyframes is fast. As an alternative, we can exploit the information about the ordering of keyframes and match against a set of n prior keyframes $\{k_{t-1}, k_{t-2}, \ldots, k_{t-n}\}$.

After this, we estimate the epipolar geometry using the pose information from the VINS algorithm, to remove the outliers and save the verified matches into the vector of correspondence descriptions *desc*. We provide more details in section 4.1.4.

Once we have at least m correspondences for a given correspondence descriptor, with m being another important parameter of the algorithm, we can triangulate a new point, which we describe in section 4.1.5.

4.1.2 Keyframe selection

The frame becomes a keyframe if it satisfies the requirements of minimal translational and rotational distances to the last frame. Translational distance is defined simply as the standard Euclidean metric between two camera centres. We define the rotational distance in the following section.

Rotational distance

In [51] a metric r on the space of rotational matrices $SO(3)$ is defined as:

$$\begin{aligned} &r : SO(3) \times SO(3) \to \langle 0, \pi), \\ &r(\mathbf{R}_1, \mathbf{R}_2) = ||\log(\mathbf{R}_1\mathbf{R}_2^T)||, \end{aligned} \tag{4.3}$$

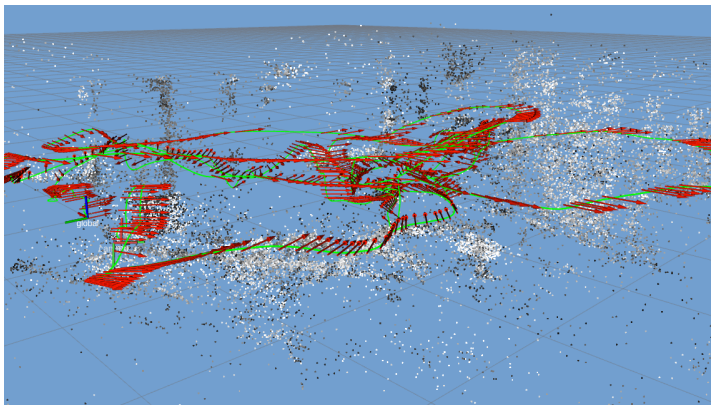


Figure 4.1: Resulting point cloud and the keyframes (red arrows). The green line symbolises the trajectory.

where log gives the vector representation of rotation derived from the Euler-Rodrigues formula as described in the section 3.3, hence the norm gives the angle θ of rotation around the axis $\mathbf{a}$. This function can be proven to be a metric and also has some desirable geometric properties that are crucial in our case, for example that two rotation matrices with similar values will also have a small value of this metric. The intuition behind this formula is that it attempts to find the amount of rotation required to align $\mathbf{R}_1$ to $\mathbf{R}_2$. For the proof, details about the computation and properties of this metric, we refer the reader to [51].

4.1.3 Keypoint detection and description

For the matching-based algorithm, we chose the Features from Accelerated Segment Test (FAST) feature detector and ORB feature descriptor, as they are efficient to compute and use while at the same are precise enough for our use. For evaluation of ORB performance, refer to [20]. Also, multiple other works have been based on ORB such as ORB-SLAM [33].

We detect features in cells of a regular $n \times n$ grid, meaning that we always attempt to find k features for each cell. We do this to make the distribution of detected points more even, attempting to detect some keypoints in every part of the image. At the same time, we do not want to dictate the points' location as then less high-quality points would be detected, deteriorating the matching quality. In practice, we opted for values $n = 4$ and $k = 100$.

FAST detector

The FAST detector was first introduced in [55] and then improved in [56]. It is based on the idea that a corner should have in its surroundings points differing in intensity. It uses the information from a Bresenham circle $C \subset \Omega$ of 16 pixels around the point of interest $\mathbf{p} \in \Omega$. It evaluates pixels in two passes. In the first pass, a less accurate but more computationally effective test is performed to filter potential candidate points. This test considers only four points on the circle, namely points in the x-axis and y-axis direction from $\mathbf{p}$. The point is considered for the second test only if for three points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in C$ of the four holds one of

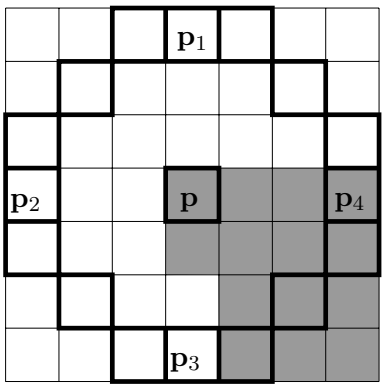


Figure 4.2: Example of a FAST corner $\mathbf{p}$, with the Bresenham's circle containing the points used in the first test.

the following conditions:

$$\begin{aligned} I(\mathbf{p}_i) &< I(\mathbf{p}) - t, \ \forall i = 1, 2, 3 \\ I(\mathbf{p}_i) &> I(\mathbf{p}) + t, \ \forall i = 1, 2, 3 \end{aligned} \tag{4.4}$$

where t is a pre-defined threshold. A typical value is 10. The candidates are then evaluated by a more precise test, where one of the above conditions must hold for at least 12 contiguous points. A simple example of the FAST corner is illustrated in figure 4.2. The grey area symbolises pixels that have a similar intensity value to $\mathbf{p}$, while white pixels have the intensity value lower at least by the FAST threshold.

FAST detector usually picks many points within close proximity. This can be remedied by using non-maximal suppression. A scoring function is introduced:

$$V(\mathbf{p}) = \sum_{\mathbf{x} \in C} |I(\mathbf{x}) - I(\mathbf{p})|. \tag{4.5}$$

For two close points, we discard one with the lower value of V. We can also sort the points using this function and pick the points with the highest score.

ORB descriptor

ORB is a feature descriptor, but part of the contribution of ORB is also improved feature point detection. This comes at a computational expense, however, which is why we decided to use plain FAST for detection and ORB only for description and matching. We will only describe the ORB descriptor, not the detection mechanism.

ORB descriptor builds on top of Binary Robust Independent Elementary Features (BRIEF) descriptor, improving its rotational invariance. The BRIEF descriptor is a binary vector of $n = 256$ intensity tests $\mathbf{t} \in \{0, 1\}^n$. Intensity test τ of points $\mathbf{p}_1, \mathbf{p}_2 \in \Omega$ in smoothed image I is defined as follows:

$$\tau(\mathbf{p}_1, \mathbf{p}_2) = \begin{cases} 1 & \text{if } I(\mathbf{p}_1) < I(\mathbf{p}_2) \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

The rotational invariance is accomplished by taking into account the orientation of the keypoints in the descriptor and by learning which tests are more valuable. Details of this can be found in the work [21].

4.1.4 Matching

Matching of descriptors is performed by a nearest-neighbour search over some descriptor distance function. As ORB feature descriptor is a binary vector $\mathbf{t} \in \{0,1\}^n$, the function used is Hamming distance, which is equal to the number of positions in which the vectors differ. We accept two descriptors as matches if they are the nearest neighbours and their distance is lower than some threshold.

Matching usually produces a considerable amount of false-positive erroneous matches. This can be reduced by a simple test introduced by D. Lowe in [19]. We perform the 2-nearest-neighbour search. It is then sensible to assume that a match should be unique. Therefore if the second nearest neighbour is too close we refuse the match as not unique enough. Let d_1, d_2 be the smallest and second smallest, respectively. We then accept the match if it holds that $d_1 < ld_2$, where $l \in (0,1)$ is a constant ratio threshold. We experimentally set $l = 0.7$.

Geometric verification

We further filter the matches using epipolar geometry, which we can directly compute by leveraging the information provided by the localisation algorithm. Namely, we construct the essential matrix from the definition (3.15). Then, for each pair of matched points $\mathbf{x}_1, \mathbf{x}_2$ in normalized image coordinates we compute the epipolar lines l_1, l_2 given by the equation (3.14). We compute a regular distance of a point from the line for points and epipolar lines in both frames and average them. Then we set threshold t and refuse a pair of matching points if the value of the average epipolar error is larger than t.

The value of this threshold depends on the quality of the localisation algorithm, as this dictates the precision of the epipolar geometry. In our case, we found values of around 0.05 to work well.

4.1.5 M-view triangulation

For point triangulation, we are using the m-view triangulation method that gives better results than regular 2-view triangulation. This method provides us with an initial estimate of the position of a point, which we further refine over time by a non-linear optimisation method described in section 4.1.6.

We have used the multi-view triangulation algorithm present in OpenVINS [41]. It goes as follows.

Suppose that we have an unknown point $\mathbf{x} \in \mathbb{R}^3$ that is observed by m cameras with their centres $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m$. We pick some arbitrary camera frame as an anchor a. We can then write the following transformations, for arbitrary camera frame i and anchor frame a:

$$\begin{aligned} {}^{i}\mathbf{x} &= \mathbf{R}_{a\to i}({}^{a}\mathbf{x} - {}^{a}\mathbf{c}_i), \\ {}^{a}\mathbf{x} &= \mathbf{R}_{i\to a}{}^{i}\mathbf{x} + {}^{a}\mathbf{c}_i. \end{aligned} \tag{4.7}$$

Also, we can represent the point coordinates within the frame of i'th camera ${}^{i}\mathbf{x}$ as the coordinates of its projection onto the image plane scaled by some factor $s_{i_\mathbf{x}}$

$$ {}^{i}\mathbf{x} = s_{i_\mathbf{x}}\mathbf{x}_{proj_i}, \tag{4.8} $$

where $\mathbf{x}_{proj} = [u_n, v_n, 1]^T$ and u_n, v_n are the normalized image coordinates of the projection, hence from our point of view we can deem them to be the image coordinates of the found correspondence. We can substitute this into the second equation of (4.7) and we get

$$ {}^{a}\mathbf{x} = s_{i_\mathbf{x}}\mathbf{R}_{i\to a}\mathbf{x}_{proj_i} + {}^{a}\mathbf{c}_i. \tag{4.9} $$

Let us name the vector $\mathbf{R}_{i\to a}\mathbf{x}_{proj_i}$ as $\mathbf{v}_i$. We define a skew-symmetric matrix $[\mathbf{v}_i]_\times$ in the same way as in equation (3.10). Because rows of the matrix are orthogonal to the vector $\mathbf{v}_i$, we can multiply (4.9) by it and get

$$ [\mathbf{v}_i]_\times {}^{a}\mathbf{x} = [\mathbf{v}_i]_\times {}^{a}\mathbf{c}_i. \tag{4.10} $$

This way we reduced the degrees of freedom by removing the parameter $s_{i_\mathbf{x}}$. We can write a linear equation

$$ \underbrace{\begin{bmatrix} \vdots \\ [\mathbf{v}_i]_\times \\ \vdots \end{bmatrix}}_{\mathbf{A}} {}^{a}\mathbf{x} = \underbrace{\begin{bmatrix} \vdots \\ [\mathbf{v}_i]_\times {}^{a}\mathbf{c}_i \\ \vdots \end{bmatrix}}_{\mathbf{b}}. \tag{4.11} $$

Because each measurement $[u_n, v_n]_i^T$ gives us two constraints on the solution, we should have enough constraints to triangulate the feature as usually we require more than 3 correspondences to compute the triangulation due to quality reasons. We can further simplify the solution process by noting that we can multiply both sides from the left by a transposed matrix of the left side:

$$ \mathbf{A}^T\mathbf{A}{}^{a}\mathbf{x} = \mathbf{A}^T\mathbf{b}. \tag{4.12} $$

This leaves us with an equation that has only a 3×3 matrix $\mathbf{A}^T\mathbf{A}$ and is easily solvable.

By solving this equation, we obtain the pose coordinates in the a coordinate frame, which we can easily transform into the global coordinate frame and get the final solution.

4.1.6 Non-linear least-squares refinement

The initial estimates obtained by the triangulation are further refined using a non-linear least-squares method. This refinement runs in a separate thread and is performed when either:

1. point is newly-triangulated or
2. a new match of some point is found and the time elapsed since the last optimisation of the point is at least one second.

The time restriction is vital to ensure that we do not waste the computing power recomputing the same points too frequently, which would lead to delays in optimisation of the newly-triangulated points.

Consider a point $\mathbf{x} \in \mathbb{R}^3$ that has a set of n matches $\mathbf{m}_i$ in normalized undistorted image coordinates with corresponding camera extrinsic matrix $\mathbf{P}_i \in \mathbb{R}^{3\times4}$ for $i = 1, \ldots, n$. We want to minimize the following function w.r.t. $\mathbf{x}$:

$$\sum_{i=1}^{n} \frac{1}{2} ||h(\mathbf{P}_i \bar{\mathbf{x}}) - \mathbf{m}_i||^2. \tag{4.13}$$

This is a non-linear least squares problem (refer section 3.5) where the non-linearity stems from the fact that the function h, converting points from homogeneous coordinates, performs division by the last coordinate. We solve a robustified version of this problem as we defined in section 3.5.2, to reduce the impact of outliers that arise here from errors in matching.

4.1.7 Point quality measures

We can try to judge the quality of a triangulated point based on a set of heuristic measures that are simple to compute. We filter points based on these metrics to try to ensure some quality of the resulting structure. However, we recommend setting soft thresholds on these metrics, as they can be overly pessimistic and thus considerably restrict the density of the structure.

The first metric we keep track of is the reprojection error. This is the metric that the whole problem chooses to optimise for. Therefore it is clear why it is useful to be keeping its value.

Next, we also measure the maximum baseline between frames that the point is visible from. The reasoning is that larger baselines provide more reliable information for triangulation and thus should lead to more accurate points' positions.

The last metric we compute is the average distance of the matches to the epipolar line (described in section 3.4). This is based on the assumption that we have an accurate estimate of the epipolar geometry that can be effectively used to judge the quality of matches.

4.2 Small-baseline tracking-based algorithm

This algorithm has an underlying principle that is rather different from the previous one, even though the fundamentals remain the same. The fundamental functionality of the algorithm is based upon a tracker that works on a frame-to-frame basis, rather than on a keyframe basis. This means that we have more matches for an individual point and, more importantly, the matching of points happens on a very small baseline. That allows us to employ a very different approach to matching that does not require computation of descriptors and their subsequent matching. Instead, we detect points in the first frame and then try to locate them within their local neighbourhood in the next frame by some similarity measures. This approach is called tracking and can generally be used for tracking the moving points and objects within a stationary video sequence. Here we assume the scene to be static; hence we instead use tracking to accomplish the same result as matching did.

Our first assumption was that this method would have to perform much more work in a given time unit, as it has to track constantly frame to frame and cannot utilise a segment

of time for some other computations. We found that this does not matter much and that it can be beneficial as it provides us with more data and also is able to compute more dense point clouds. Especially in areas with weak texture and not many distinctive features, this algorithm can perform relatively well, leveraging the locality of tracking. For more discussion and data, please see the performance evaluation in the experiments' chapter 6.4.

4.2.1 Algorithm's structure

The general structure is very similar to that of the previous algorithm, except that now we keep information about all the frames. Despite processing all frames, the algorithm is able to achieve a framerate of around 25 – 30 FPS.

In the first frames we detect some points using the FAST detector that we described in section 4.1.3, that we then track to next frames. As a tracking algorithm, we use the Kanade–Lucas–Tomasi (KLT) tracker, which we describe in more detail in section 4.2.2.

The point is triangulated once we have tracked it through a sufficient amount of frames. This is similar to the matching-based algorithm, although here we triangulate when we have a higher number of matches to cope with small-baseline-related issues by promoting data redundancy in the initial triangulation. The triangulation algorithm itself is the same as in the matching algorithm, see section 4.1.5. Also, the non-linear refinement mechanism is identical, see section 4.1.6.

Furthermore, we developed a mechanism of detecting outlier points by operations on the local neighbourhood of a point which is defined by a 2D Delaunay triangulation. This can also be used to perform some light point cloud densification. We describe this in section 4.2.3.

This algorithm also performs keyframe selection based on the same criteria as the previous, but keyframes are intended mainly for the post-processing step and as means of reducing redundancy in the Delaunay computation, not for the tracking itself.

4.2.2 Kanade–Lucas–Tomasi (KLT) tracker

KLT tracker is an iterative algorithm that attempts to find for points detected in the first image their correspondences in the second image frame. It was introduced in the two papers [57, 58] by Kanade, Lucas and Tomasi, hence its name. The correspondences are found by computing a motion vector between the two frames. An assumption is made that the camera movement between the frames is small; hence the tracker is best suited for video sequences. The initial points can be detected by multiple methods. In this work, we used the FAST algorithm that we described in section 4.1.3.

Let $T : D \to \mathbb{R}, D \subset \Omega$ be a patch template in the first frame. This means that it is a section of the first image. The patch is transformed to second frame I by some transformation $W(\mathbf{x}; \mathbf{p})$ where $\mathbf{p}$ is a vector of its parameters. We want to estimate the optimal parameters $\hat{\mathbf{p}}$ by approximately minimizing the following functional:

$$\sum_{\mathbf{x} \in D} [I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2. \tag{4.14}$$

This corresponds to the minimisation of the visual difference between the original and the transformed patch. The resulting transformation can then be used to compute the position of the point $\mathbf{x}_i \in D$ in the second frame, $\mathbf{x}_{i+1} = W(\mathbf{x}_i; \hat{\mathbf{p}})$. The function W can be simple translation, rigid motion, affine or projective transformation. We use the affine transformation, but we will derive a general KLT algorithm. Initial estimate of $\mathbf{p}$ is assumed to be known and we try to estimate its change $\Delta\mathbf{p}$:

$$\Delta\mathbf{p} = \underset{\Delta\mathbf{p}}{\operatorname{argmin}} \sum_{\mathbf{x}\in D} [I(W(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2. \tag{4.15}$$

Let us linearise the functional $I(W(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p}))$ by a first-order Taylor series approximation:

$$\sum_{\mathbf{x}\in D} [I(W(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x})]^2. \tag{4.16}$$

Here the ∇I represents the image gradient evaluated at $W(\mathbf{x}; \mathbf{p})$. We can differentiate the above term and set it equal to zero and we get the value of $\Delta\mathbf{p}$ as

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}\in D} \Big[\nabla I \frac{\partial W}{\partial p}\Big]^T [T(\mathbf{x}) - I(W(\mathbf{x}; \mathbf{p}))], \tag{4.17}$$

where $\mathbf{H} \in \mathbb{R}^{n\times n}$ is the Gauss-Newton approximation of the Hessian matrix:

$$\mathbf{H} = \sum_{\mathbf{x}\in D} \Big[\nabla I \frac{\partial W}{\partial p}\Big]^T \Big[\nabla I \frac{\partial W}{\partial p}\Big]. \tag{4.18}$$

Now the whole algorithm can be written as a simple iterative procedure 2.

Algorithm 2: KLT tracker

```
Given initial p, ε, T, I and W;
while ||Δp|| > ε do
    Compute Δp according to (4.17);
    p := p + Δp;
end
```

Essentially, the solution to the KLT is produced by solving a non-linear least-squares problem by a specific iterative method.

There is an inherent trade-off between the accuracy and robustness of the tracker. Intuitively, the smaller the patch window, the more accurate results we will get. With the growing size of the window, we lose the local details as their influence on the functional becomes small. On the other hand, bigger windows handle better larger movements. Therefore, the KLT tracker is usually implemented using an image pyramid that is processed in a coarse-to-fine manner. We begin by estimating the parameters at the deepest image, which corresponds to the smallest image with a low number of details. The estimates are then propagated through the pyramid to account for the finer details. This method was introduced in [59].

For a very detailed account of the theory, practicalities and evaluation of the KLT tracker, we refer the reader to [60].

Management of points

New points must be continuously detected, as some may move out of the view, become occluded or lost by the tracker. Also, some points might move too close during tracking. We can pick just one point out of the group of close points. This is accomplished by splitting the image into a grid and enforcing that there must be at most a single point within each grid cell.

We, therefore, try to sample new points proportionally to the number of points needed to reach our desired number of points. The same gridded detection mechanism is employed as in the matching-based algorithm. For details refer section 4.1.3. We also tried to detect points proportionally to the number of points needed in individual cells. However, we could not measure any major improvements; hence we decided to use the simpler method. To promote redundancy, we try to detect some constant amount of points even when we have more points than we specified. This does not lead to unlimited growth, as in reality we almost always drop enough points.

4.2.3 Delaunay mesh

A Delaunay triangulation of the set of points in the image has proven to be a useful concept in the context of filling gaps and control of points' quality. It gives us useful information about a local neighbourhood of the point.

We will first describe the Delaunay triangulation and then outline its uses in this work. The idea to use Delaunay triangulation comes from [38].

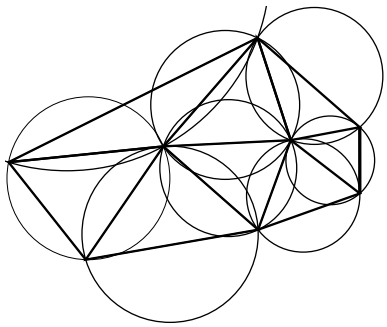


Figure 4.3: Example of a Delaunay triangulation of a set of points, with the cicumcircles.

Delaunay triangulation

The process of triangulation here means, in the geometric sense, a subdivision of some planar object into triangles. This is not to be confused with the concept of triangulation as used by the computer vision community that we have mentioned extensively throughout this work.

We will not define triangulation fully formally, as this would be out of the scope of this work. But informally, for a triangulation of a non-collinear set of points $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}, \mathbf{p}_i \in \mathbb{R}^2$ must hold:

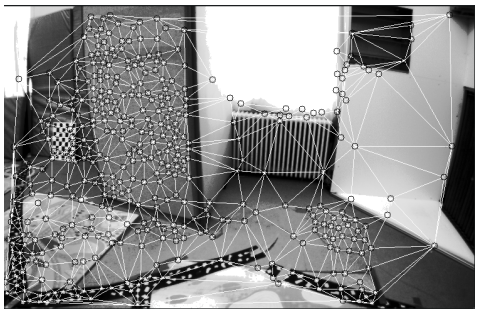


Figure 4.4: The Delaunay triangulation in the runtime of the algorithm.

1. Every point $\mathbf{p}_i$ is a vertex of at least one triangle.
2. Triangles do not intersect with each other but may share a vertex or edge.
3. Triangles form a convex polygon.

Delaunay triangulation is then such a triangulation that the circumcircle of every triangle does not contain any $\mathbf{p} \in P$. The circumcircle of a triangle is a circle that passes through all of its three vertices. It is uniquely determined by the vertices. An example of a Delaunay triangulation with visualised circumcircles can be seen in figure 4.3 and an example of the Delaunay triangulation in the algorithm is in figure 4.4.

Outlier removal

We propose a novel method for the removal of outlier points that is based on the Delaunay triangulation connectivity graph.

Consider a point $\mathbf{x} \in \mathbb{R}^3$ with a correspondence in some camera frame c. Let Δ be a set of points whose correspondences in c share an edge with the correspondence of $\mathbf{x}$ in the Delaunay triangulation.

We define a simple outlier-rejection scheme, which uses a least-squares interpolation of a plane passing through the depth points. For reference, we defined depth of point in section 3.2.3. Let us define a linear plane depth function:

$$D(\bar{\mathbf{a}}; \mathbf{p}) = \mathbf{p}^T \bar{\mathbf{a}}, \tag{4.19}$$

where $\bar{\mathbf{a}} \in \mathbb{R}^3$ is a point correspondence in homogeneous normalized image coordinates.

We solve two least squares problems, one with the correspondence of $\mathbf{x}$ included and one without it, and obtain two parameter vectors $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{R}^3$:

$$\begin{aligned} \mathbf{p_1} &= \underset{\mathbf{p_1}}{\operatorname{argmin}} \sum_{\mathbf{y} \in \Delta} (D(\bar{\mathbf{a}}_\mathbf{y}; \mathbf{p_1}) - d_\mathbf{y})^2, \\ \mathbf{p_2} &= \underset{\mathbf{p_2}}{\operatorname{argmin}} \sum_{\mathbf{y} \in \Delta \cup \{\mathbf{x}\}} (D(\bar{\mathbf{a}}_\mathbf{y}; \mathbf{p_2}) - d_\mathbf{y})^2, \end{aligned} \tag{4.20}$$

where $\bar{\mathbf{a}}_\mathbf{y}$ is the correspondence in c of the point $\mathbf{y}$ and $d_\mathbf{y}$ is its depth in c. These are linear least-squares problems that we can quickly solve by pseudo-inversion. We then consider the point to be an outlier if it holds that $||p_1 - p_2|| > t$, where t is a threshold.

This approach assumes that the surrounding depths of near points can be either well approximated by some plane and that the large difference between $\mathbf{p_1}$ and $\mathbf{p_2}$ is indeed caused by the outlying point $\mathbf{x}$. This is a gross oversimplification, but in man-made environments that are mostly planar, it enables us to deal with some of the outliers. Especially when we limit the definition of D by the intensity and distance thresholding. This means that we take into account only neighbouring points with similar intensities and low pixel distance to the correspondence $\mathbf{a_x}$. In practice, we set $t = 0.7$, although we note that more evaluation of settings of this threshold in different environments is needed.

This approach was not utilised in the matching-based algorithm, as there the detected points were too sparse; therefore, this method could not work reliably under such circumstances.

Densification

We can densify the point cloud by assuming the planar model of a single triangle, as in the previous section. We sample the 2D triangle and project the new points by using the interpolated depth. The plane has a closed-form solution, as 3 points define a plane unambiguously, provided they are not on a single line. Assume we have three points forming the Delaunay triangle, $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{R}^3$, where their last coordinate is the depth. The normal vector $\mathbf{n}$ of the plane with normal equation $\mathbf{x}^T\mathbf{n} = l$ can then be computed as:

$$\mathbf{n} = (\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1). \tag{4.21}$$

To compute the affine constant l, we can substitute one of the points into the plane equation.

We sample the points from the triangle on a regular grid rotated so that it matches the direction of one of the edges of the triangle. For simplicity, we do this in the normalised image coordinates, although we believe that better approaches might exist and could improve the performance of this algorithm. The dimension of the grid we set to 0.05, which would correspond to 5 % of pixels in the horizontal or vertical directions if the grid was aligned to them.

To avoid redundancy, we perform the densification procedure only in the keyframes that are determined in the same way as in the matching-based algorithm.

Importantly, suitable triangles for this operation must be chosen, as some would lead to many erroneous points. Firstly, we refuse triangles defined by points that are too far apart from one another, as it is not probable that this triangle is a good approximation of the space between the two points. We accomplish this by setting a threshold on the maximum length of an edge of the triangle.

Further, we limit the maximum photometric difference between the triangle points. This is a common assumption employed in the depth estimation problems: that when points are close in the image and have a similar pixel intensity value, they will probably be close in reality, and their surroundings can be linearly approximated. Naturally, this is a simplification that may not hold under many circumstances, e.g. the crown of a tree. In our experiments, we set this maximum distance to be 0.2 m, as this corresponds to the value of the resolution of the occupancy grid, which we describe in the following section 4.3.

Let the pixel intensity values of the vertices of the triangle be denoted as i_1, i_2, i_3. We then accept the triangle only if the following condition holds:

$$\mu - a\sigma \leq i_j \leq \mu + a\sigma, \forall j = 1, 2, 3, \tag{4.22}$$

where μ is the average of the pixel intensity values, σ is its standard deviation and a is a parameter influencing the strictness of the condition. In our experiments, we chose $a = 1.5$.

We provide the coordinate of the filled points on demand by computing it using the triangle interpolation of the depth and its conversion to global coordinates. This means that these points can move over time, as the non-linear least-squares refinement step optimises the points that define the triangle or the pose of the frame. We do not attempt to match the filled points in other frames, although we note that this can be an interesting direction of improvements.

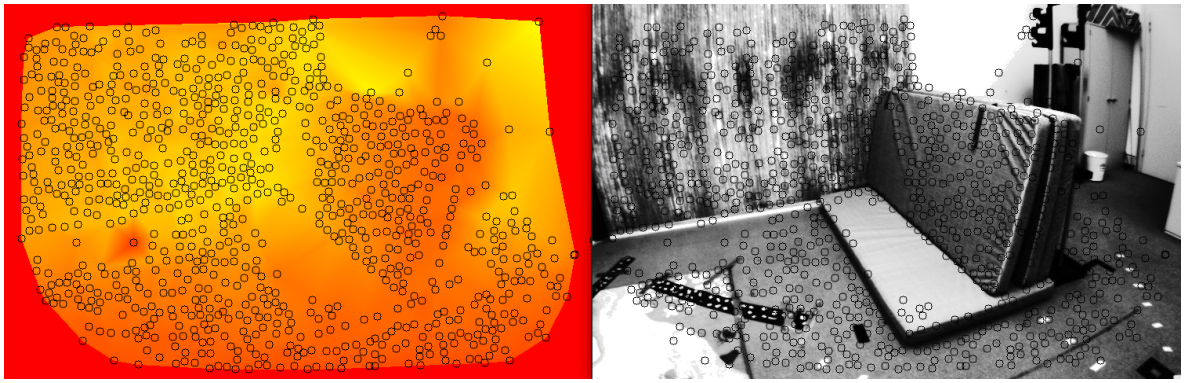


Figure 4.5: Depth map created by the linear interpolation over the Delaunay triangulation. The orange end of the colourmap corresponds to lower depth. The visible triangulated points, upon which the triangulation is based are also displayed.

We can use this method to produce full depth maps, as we show in figure 4.5. However, we did not use this in the final algorithm, as the depth map would require some additional optimisation to smooth out errors, such as in the paper [38]. We also note that the whole depth map is largely redundant for the purpose of autonomous navigation. Furthermore, it is relatively computationally expensive when performed on a CPU.

4.3 Interface with the navigation algorithms

Despite our method producing substantially denser environment representation in the form of a point cloud than the VINS algorithms, it is still too sparse for a direct application of navigation algorithms, as they need to see the gaps between the points closed. Because of this, we feed the point information into some other algorithm that computes representation that can be directly used by existing planning techniques. We used the OctoMap [61] method, which was chosen as it is already used within the MRS system for processing the data generated by LIDAR sensors, which are also point cloud data. That means there are navigation algorithms already implemented that use the OctoMap and could therefore be easily employed with this method.

The OctoMap works by constructing an occupancy grid representation of the 3D space, where each cell can either represent empty or occupied space. It probabilistically integrates the information about point density from the point cloud to update the knowledge about occupied space.

In essence, the probability of being occupied and free is kept in each cell in the space and is manipulated by observations. When a point is observed, the occupation probability of the cell containing the said point is increased. Importantly, a raycasting operation from the sensor (in our case camera) to the point is performed, and the probability of being occupied is decreased for all the grid cells along this ray.

An important parameter is the resolution of the octree, which is the size of the edge of its leaf cell, or, equivalently, to the size of the occupancy grid cell. We use the same value that is already being used at MRS for the LIDAR scans, which is 0.2 m.

Chapter 5

Offline Post-Processing of the 3D Structure

Contents

In this chapter, we describe an offline post-processing step that seeks to improve the quality of the resulting 3D structure. It consists of three main parts: full bundle adjustment, loop closure and outlier rejection. As this step is performed offline, we can utilise some computationally heavy components that could not be used in the online setting.

5.1 Full Bundle Adjustment

Consider a scene graph of n points $\mathbf{x}_i \in \mathbb{R}^3$ and k camera intrinsic matrices $\mathbf{P}_l \in \mathbb{R}^{3\times4}$. Each of these points has a set of matches in undistorted normalized image coordinates $M_i = \{{}^{j_1}\mathbf{m}_1, \ldots, {}^{j_m}\mathbf{m}_m\}$ in m camera frames $j_1, \ldots, j_m$.

The full BA is then defined as a joint optimisation problem of point positions $\mathbf{x}$ and camera extrinsics $\mathbf{P}$ over the scene graph, in which we minimize the following function:

$$\sum_{i=1}^{n} \sum_{{}^{j}\mathbf{m}\in M_i} ||h(\mathbf{P}_j \bar{\mathbf{x}}_i) - {}^{j}\mathbf{m}||^2. \tag{5.1}$$

This is again a non-linear least-squares problem (for details refer section 3.5). Note that, in general, the intrinsic parameters can also be optimised, which is not necessary in our case as we assume them to be known. The BA is a more general version of the structure refinement problem we were solving online 4.1.6. However, the problem becomes more complex and computationally expensive, which prohibits us from solving it in the real-time setting, especially considering that we aim for a large number of points.

We solve the full BA problem for just a subsection of frames, called the keyframes. The way we determine those is described in section 4.1.2. By fixing the rotation or translation part of the extrinsic matrix $\mathbf{P}$, we can further obtain two simpler problems that allow the use of the full scene graph. We will compare those approaches in the experimental section 6.4.4.

We note that the BA problem has a certain specific sparse structure that can be exploited for a more effective solution of the linear squares sub-problem 3.5.1. This stems from the fact that in most scenes, points can only be observed from a small subset of frames. We will not describe the details of these methods in this work. We will only note that we use the Schur complement method, and for a more detailed treatment of the topic, we refer the reader to [27].

The rotational parts we parametrise as rotation vectors because they have better properties for optimisation. We described this parametrisation in section 3.3. Namely, they have the minimum number of parameters needed to represent rotations and, also, they do not have to be normalised in any way during the optimisation, as a rotation matrix would have to be.

5.2 Loop closure

In order to provide the BA optimisation with more information about the structure, we employ a simple loop closure scheme. The problem of loop closure is to introduce new connections in the scene graph, capturing the fact that two points observed in two frames that are far apart in time are the same point in the real world. This establishes more constraints for the optimisation, potentially enhancing its results.

Our loop closure scheme is based on the assumption that the employed localisation system is precise, meaning that when two frames are close in the map of the environment, they are also close in the real world. We thus examine frames i, j that satisfy the following conditions:

$$d(i,j) < d_{thr}, \tag{5.2}$$

$$r(i,j) < r_{thr}, \tag{5.3}$$

$$i \lll j, \tag{5.4}$$

where $d(i,j)$ is translational distance between the two frames and $r(i,j)$ is the rotational distance between the two frames that we defined in 4.1.2. The conditions (5.2), (5.3) enforce the frames to be close from each other, therefore making it more probable that they will view a common part of the scene. The third condition (5.4) ensures that the indexes of the two frames are far apart from each other, which effectively enforces a temporal difference between the two frames. This is due to the fact that close frames with similar ids have a higher chance of having already correctly tracked points of the same 3D structure.

We then perform the matching of triangulated points observed from these frames by computing SIFT descriptors and using those. The SIFT descriptors could not be used during the online part of the algorithm due to their high computational cost. However, here we are not limited by strict resource constraints, and SIFT descriptors are beneficial, as they exhibit a much higher rate of correct matches compared to ORB descriptors, especially under a big change of scale and rotation [20].

The matching is performed in a similar fashion to 4.1.4. As SIFT is not a binary descriptor as ORB, regular Euclidean distance is used to determine similarities between descriptors. For a detailed description of the SIFT algorithm, we refer the reader to [19]. When a match is established, we then merge the two points.

5.3 Outlier rejection with a SOR filter

In addition to the online rejection method discussed in section 4.2.3, we also implement a rejection method that takes into account the local structure of the point cloud.

We use an approach called the statistical outlier removal (SOR) filter from the Point CLoud Library (PCL) [62]. It considers k-nearest neighbours of each point $\mathbf{x} \in \mathbb{R}^3$. A point $\mathbf{y}$ is considered to be an inlier if the following condition holds:

$$\mu - a\sigma \leq ||\mathbf{x} - \mathbf{y}|| \leq \mu + a\sigma, \tag{5.5}$$

where μ is the mean distance of the k-nearest neighbours to the point $\mathbf{x}$, σ is the standard deviation of these distances, and a is a user defined constant that can be used to control how strict the filter is. With this filter, we can remove many of the clear outliers. For our experiments we set the $k = 10$ and $a = 1$.

We also considered using a filter measuring the distance of the point to other points to remove some outliers that are distant from the rest of the point cloud. However, by using the k-nearest neighbours distance filter, we removed almost all of the outlier points that we would target by such a filter.

Chapter 6

Experimental Evaluation

Contents

We performed a wide range of tests of the implemented algorithms. The EuRoC datasets were first used to compare and evaluate the algorithms' performance under different difficulty levels. The performance could be measured very precisely, as the dataset comes with a ground-truth point cloud of the scene obtained by a high-resolution LIDAR scan.

Importantly, when we talk about OpenVINS in the evaluation section, we mean its monocular version as that is the problem we are solving.

More visual content from the experiments, including the resulting point clouds, can be found on the website accompanying this thesis[1].

6.1 Implementation and used libraries

The code is written in the C++ programming language, which is a standard choice in robotics and computer vision because of its performance optimisation and library support.

Robot Operating System (ROS) is a widely used open-source library, providing an abstraction for effective development of algorithms for the operation of robots, ranging from sensor controllers all the way to high-level path-planning algorithms [63]. One of the most important features is that it enforces the code to be organised in task-oriented programs

[1] `http://mrs.felk.cvut.cz/melecky2021thesis`

called nodes and enables to structure these nodes into complex structures by inter-node communication. The code-base is maintained, well-documented, and a wide community exists around it, making it comparably simple to get into. Another feature we will use to our advantage in this work is the Rviz visualisation system that can be used as a simple graphical output of the algorithm, streaming information in real time.

For some computer vision methods, most notably the KLT tracker and descriptors, we use OpenCV [64], which is an open-source library for computer vision. A wide range of computer vision methods is implemented in it. We can also quite easily use OpenCV implementations of algorithms as a baseline to test our implementations against.

For operations with matrices and linear algebra in general, we utilise the Eigen library [65].

We use the Ceres library [54] for solving most of the non-linear least squares problems that we discussed in the theoretical parts of the work [54].

For operations on point clouds, such as the SOR filter, we leveraged the Point CLoud Library (PCL) [62]. For visualisations of the mean distance to reference (MDR) error in the figures, we use the CloudCompare software[2].

6.2 Parametrization of the algorithms

	Parameter	Value
Keyframes	Minimum distance between keyframes	0.03 m
	Minimum rotation distance between keyframes	0.008 rad
	Length of sliding window	4
Matching	ORB scale factor	1.2
	FAST threshold	8 px
	Lowe ratio	0.7
	Threshold on epipolar distance	0.05 m
	Grid dimension	4
	Maximum amount of features per grid cell	100
Triangulation and refinement	Amount of matches required for triangulation	4
	Huber loss threshold	0.1
	Maximum movement caused by refinement	5 m

Table 6.1: Parametrization of the matching-based algorithm.

In the tables 6.1 and 6.2, we provide a concise reference of the most important parameters of both algorithms and their values. In the following tests, we assume the parameters' values to be set as described here unless we specify otherwise. The parameters' values were

[2] `https://www.danielgm.net/cc/`

	Parameter	Value
Tracking	FAST threshold	10 px
	Threshold on epipolar distance	0.05 m
	Grid dimension	4
	Maximum amount of features per grid cell	300
Triangulation and refinement	Amount of matches required for triangulation	4
	Huber loss threshold	0.1
	Maximum movement caused by refinement	5 m
Outlier filtering	Delaunay plane difference	0.7
	Maximum epipolar violation	0.1 m
	Minimum length of maximum baseline	0.1 m
	Maximum reprojection error	0.03 m

Table 6.2: Parametrization of the tracking-based algorithm.

chosen empirically. In this section, we describe some of the reasoning behind their particular values.

In the matching-based algorithm, important parameters are those influencing the selection of keyframes, as outlined in section 4.1.2. The general idea of keyframes is to keep only important information and discard the rest. However, in the end, we found that some redundancy in information is useful in real-world conditions, and therefore, we set the values of distance thresholds to be fairly low.

The epipolar distance threshold was chosen by a visual inspection of the number of outliers in the matching data.

The grid dimension parameter allows forcing the FAST detector to pick points more evenly. However, a too detailed grid leads to a high count of poor features that are hard to match.

We tried to achieve the maximum amount of features per grid cell as possible, as we aim for denser point clouds. There is an inherent trade-off between the number of triangulated points and their quality, which is why we employ the filtering approaches. We argue that it is better to keep more points at the price of gross outliers, as for navigation purposes, it is vital to notice an obstacle early, and false obstacles do not cause too much harm unless there are too many. Moreover, we are able to remove many of the outliers in the offline post-processing step. In the tracking algorithm, we could afford more features per grid cell, as the KLT tracker can compute their matches efficiently. On the other hand, in the matching algorithm, further increase of the limit did not improve the matching results as the matching procedure could not effectively match the low-quality feature points.

We decided not to use the heuristic filtering of points in the matching-based algorithm, because we would lower the amount of triangulated points too much. Moreover, we do not have enough points for the Delaunay filtering.

In the non-linear least-squares refinement, we also introduced a threshold on the distance between point's coordinates before and after the refinement as a means of detecting an unstable, hence likely low-quality point.

6.3 Map evaluation metrics

In this section, we present the main metrics used in the evaluation of our algorithms' performance. We use two metrics measuring the quality of the point clouds and two complementary metrics for evaluation of the resulting occupancy grids.

Apart from these metrics, we also evaluated some other performance measures such as number of points, number of points visible from a frame or mean reprojection error (MRE).

6.3.1 Mean distance to reference (MDR)

The main metric for point cloud evaluation is the distance of a point from the constructed point cloud to its nearest neighbour in the reference point cloud. The main limitation of this metric is that it requires a reference point cloud of the environment to be known, which often is not the case. Also, the reference point usually is subject to some measurement error of its own. However, we use point clouds produced by a stationary high-resolution 3D scanner (e.g. LIDAR), and in this case the error of the scanner is negligible compared to our errors. We can thus safely use the point cloud as a reference.

Another consideration is that the coordinate systems of the localisation system on board of the UAV must be matched to that of the 3D scanner. For this, we make use of the trajectory alignment by yaw-only rigid body transformation algorithm provided by the OpenVINS and described in [66]. This algorithm requires the ground-truth trajectory to be known, which is the case in the EuRoC Micro Aerial Vehicle (MAV) datasets.

We also note that this metric assumes that the nearest neighbour of the given point in the reference cloud is the same point in real world. This might not be the case, however for the LIDAR scans, it is a good approximation as they are comparatively dense.

6.3.2 Mean map entropy (MME)

When a reference point cloud of appropriate quality is not available, we must rely on other methods for establishing the map quality. In these methods, we assume that the map of the world should be sufficiently "sharp", meaning that a large variation in relatively close points is a product of noise. This holds true especially in man-made environments that are largely planar and points close within some radius tend to lay on the same plane and are regularly spaced. On the other hand, natural environments such as grass would produce sets of points that will be much more scattered. Here we define one such method, called the mean map entropy (MME).

In [67], the entropy of a point $\mathbf{x} \in \mathbb{R}^3$ is defined as:

$$H(\mathbf{x}) = \frac{1}{2} \ln[\det(2\pi e \Sigma(\mathbf{x}; r))], \tag{6.1}$$

where $\Sigma(\mathbf{x}; r)$ is the sample covariance of the map points within the radius r from the point $\mathbf{x}$. In our evaluation we set $r = 0.6$m.

The mean map entropy is then defined as the average of the point entropies over all of the map points.

The authors of [67] found a correlation between the nearest-neighbour distance to the reference point cloud and the mean map entropy, which is very useful to reasoning about the quality of the solution when the reference 3D structure is not available.

6.3.3 Octomap precision

When the reference 3D structure is available, we also evaluate the quality of the constructed occupancy grid. After running the algorithms, we align the reference point cloud to the frame of the VINS by utilising the method of [66]. We then create an occupancy grid by labelling the cells that contain some points as occupied. Using this reference occupancy grid, we define two metrics for evaluation of the occupancy grid. We say that a cell is correctly classified as occupied if it is marked as occupied in both occupancy grids. Firstly, we define the true positive rate as:

$$TP = \frac{\text{number of cells correctly classified as occupied}}{\text{total number of cells classified as occupied}}. \tag{6.2}$$

As a second complementary metric, we define the coverage as:

$$Cov = \frac{\text{number of cells correctly classified as occupied}}{\text{total number of occupied cells in the reference}}. \tag{6.3}$$

These two metrics differ in the denominator and express both the rate at which we classify correctly and what percentage of occupied cells we find.

6.4 EuRoC MAV datasets

The EuRoC MAV datasets [6] were the most valuable source for our tests, as they are widely used in the community, provide all the necessary data for simulating and evaluating the VINS algorithms, such as synchronised images, IMU measurements and ground-truth trajectory. Crucially for our use case, the ground-truth LIDAR scan of the environment is provided in some of the datasets as well.

We test our algorithms mainly on the Vicon Room datasets, as they provide an accurate 3D structure. The datasets are recorded within two rooms and capture different flights through these rooms. Each of the flights has different difficulty for both the SLAM and the 3D reconstruction algorithms. The differences in difficulty come from the variations in speeds and more dynamic manoeuvres. In table 6.3, we include a subsection of the table provided by the authors of the datasets in [6], that contains important parameters of the datasets: length of trajectory l, duration of the flight t, average velocity v, average angular velocity ω and a short description. We note that, for notational convenience, we took the liberty in renaming the original Difficult level to Hard.

Name	l [m]	t [s]	v [m s^{-1}]	ω [rad s^{-1}]	**Note**
V1 easy	58.6	144	0.41	0.28	slow motion, bright scene
V1 medium	75.9	83.5	0.91	0.56	fast motion, bright scene
V1 hard	79.0	105	0.75	0.62	fast motion, motion blur
V2 easy	36.5	112	0.33	0.28	slow motion, bright scene
V2 medium	83.3	115	0.72	0.59	fast motion, bright scene
V2 hard	86.1	115	0.75	0.66	fast motion, motion blur

Table 6.3: Details of the datasets.

Importantly for our algorithms, the images are captured at a rate of 20 FPS by a grayscale camera with a global shutter and resolution of 752×480.

The algorithms were run on a laptop with a quad-core Intel i5–8250U CPU. The VINS algorithm, in theory, requires a single thread, while both of our algorithms need two threads, one for tracking or matching and the second one for the optimisation of the 3D structure. In reality, the algorithms required well under half of the processing power, which in practice should allow for a sufficient resources for other tasks that must be performed on a UAV, such as path planning or motion control. We further discuss the resources required for computation in section 6.4.2.

6.4.1 Comparison of the tracking-based and matching-based algorithms

Level	**Alg.**	**MDR** [m]	**MME**	**MRE** [mm]	n [$\times 10^3$]	n_k	TP	Cov
Easy	match.	0.188 ± 0.235	-0.797 ± 0.653	3.330 ± 7.671	≈ 28	86 ± 75	0.239	0.059
	track.	0.149 ± 0.209	-0.447 ± 0.523	3.910 ± 6.102	≈ 58	578 ± 233	0.246	0.129
Medium	match.	0.194 ± 0.281	-0.773 ± 0.789	4.061 ± 12.31	≈ 21	47 ± 40	0.189	0.047
	track.	0.147 ± 0.222	-0.494 ± 0.713	7.407 ± 9.523	≈ 35	339 ± 195	0.229	0.141
Hard	match.	0.236 ± 0.355	-0.865 ± 0.910	4.530 ± 8.622	≈ 11	18 ± 23	0.414	0.081
	track.	0.214 ± 0.359	-0.344 ± 0.709	7.274 ± 8.889	≈ 35	249 ± 203	0.325	0.247

Table 6.4: Evaluation on the dataset Vicon Room 1.

In tables 6.4 and 6.5, we provide the results of an evaluation of the algorithms on the Vicon Room 1 dataset and Vicon Room 2 dataset, respectively. We evaluate the point cloud quality metrics outlined in 6.3, namely the mean distance to reference cloud (MDR) and mean map entropy (MME). Furthermore, we provide the values of metrics for the evaluation of the occupancy grids, the true positive rate TP and the coverage Cov. Also, we included some other important metrics that add some relevant information. Mean reprojection error (MRE) is added because it roughly corresponds to the metric for which the BA optimises. We also provide information about the count of points n in the resulting point cloud and the mean count of triangulated points visible from keyframes n_k. The last metric is especially relevant

Level	Alg.	MDR [m]	MME	MRE [mm]	n [$\times 10^3$]	n_k	TP	Cov
Easy	match.	0.189 ± 0.338	-0.744 ± 0.834	3.359 ± 9.221	≈ 16	72 ± 62	0.123	0.031
	track.	0.165 ± 0.281	-0.399 ± 0.705	4.407 ± 7.473	≈ 35	410 ± 185	0.126	0.081
Medium	match.	0.174 ± 0.269	-0.841 ± 0.868	3.324 ± 6.881	≈ 17	31 ± 31	0.133	0.033
	track.	0.155 ± 0.235	-0.463 ± 0.717	7.458 ± 9.084	≈ 37	276 ± 165	0.130	0.095
Hard	match.	0.236 ± 0.335	-0.896 ± 1.007	4.685 ± 10.98	≈ 11	21 ± 27	0.110	0.021
	track.	0.241 ± 0.329	-0.339 ± 0.810	10.72 ± 11.16	≈ 22	164 ± 154	0.118	0.092

Table 6.5: Evaluation on the dataset Vicon Room 2.

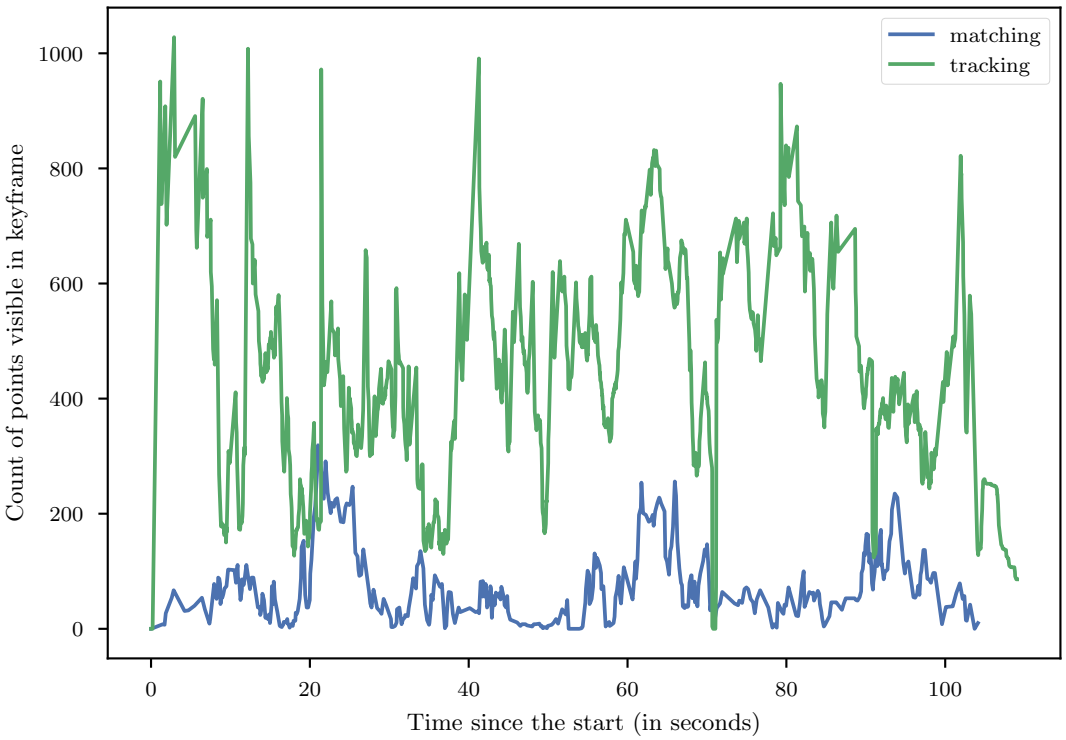
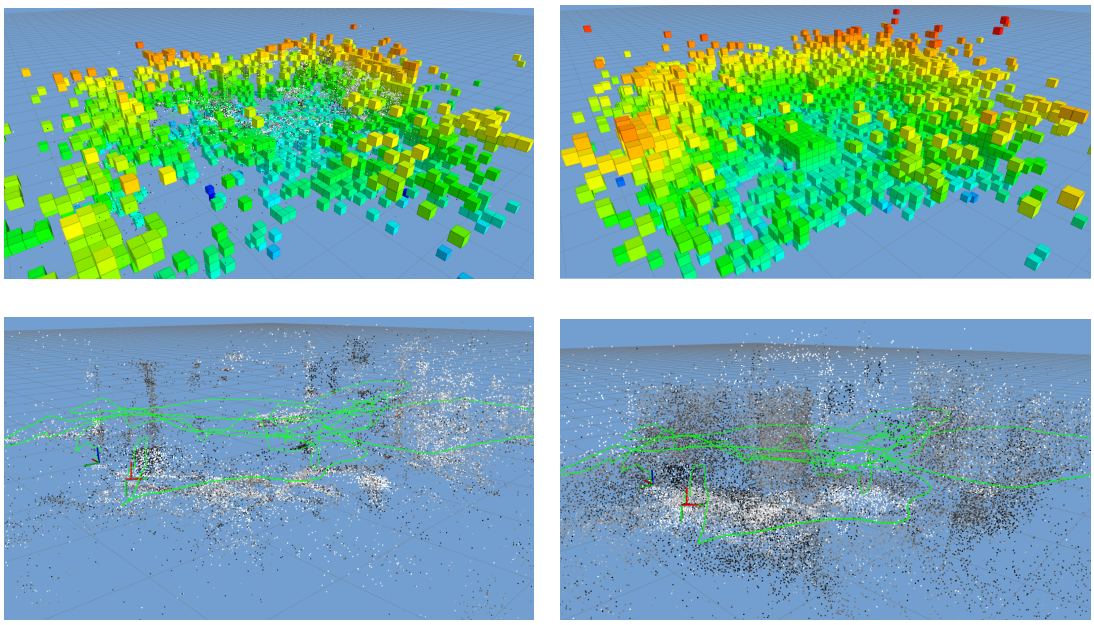


Figure 6.1: Comparison of the count of visible triangulated points from the last keyframe between the matching and tracking algorithm on the Vicon room 2 easy dataset.

for the navigation algorithms, as new obstacles must be detected quickly. Where relevant, we accompany the metrics by their standard deviation.

The time complexity of the algorithms cannot be easily compared due to the difference in approaches. The matching and triangulation time of the matching-based algorithm is typically around 40 milliseconds, while the tracking and triangulation time of the tracking-based algorithm is around 27 milliseconds. In section 6.4.2 we provide a more detailed timing analysis of the tracking-based algorithm.



(a) Matching algorithm.

(b) Tracking algorithm.

Figure 6.2: Visual comparison of the point clouds and occupancy grids produced by the proposed algorithms. In the case of the occupancy grids, the colour visualises the z coordinate.

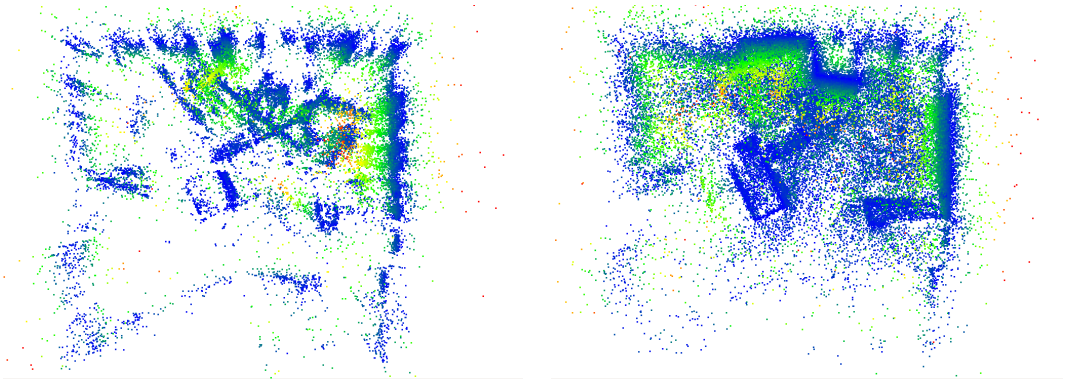
The results clearly show that the tracking-based algorithm outperforms the matching-based algorithm in most of the criteria. It is able to produce significantly more points while achieving better scores in the quality metrics, such as MDR or TP.

In the case of the true positive rate TP it is sometimes outperformed by the matching algorithm, which is not surprising, as it triangulates fewer points. However, when it comes to the coverage Cov, it is significantly higher for the tracking-based algorithm. The low number of points is a possible cause of the worse performance in the MRE of the matching-based algorithm.

It can be noted that, in general, the scores on these two occupancy grid metrics are relatively low. The low TP metric can be explained by the MDR metric, whose value is around 20 centimetres, with a standard deviation of around 30 centimetres. The resolution of the OctoMap is 20 centimetres. This means that relatively many points are further than 20 centimetres from the reference; therefore they produce many falsely occupied areas. We could limit this effect by decreasing the sensitivity of the OctoMap and by requesting higher quality of points, which can be done, for example, as was described in section 4.1.7. However, we believe that this would have a negative impact on autonomous navigation.

The low Cov can be partially explained by this, meaning that we detect some objects but place them in a slightly different place in the world, e.g. closer to the UAV. However, at the same time, the reference contains data about parts of the structure that the camera did not observe during the flight, such as the ceiling and some other parts of the room.

The coverage metric Cov is also significantly lower on the Vicon room 2 dataset. We believe that this is the case because this dataset has fewer textured areas, from which we can obtain reliable feature points.



(a) Matching algorithm. (b) Tracking algorithm.

Figure 6.3: Visual comparison of the point cloud MDR produced by the two algorithms on the dataset V1 easy. The colour scales of these visualisations match.

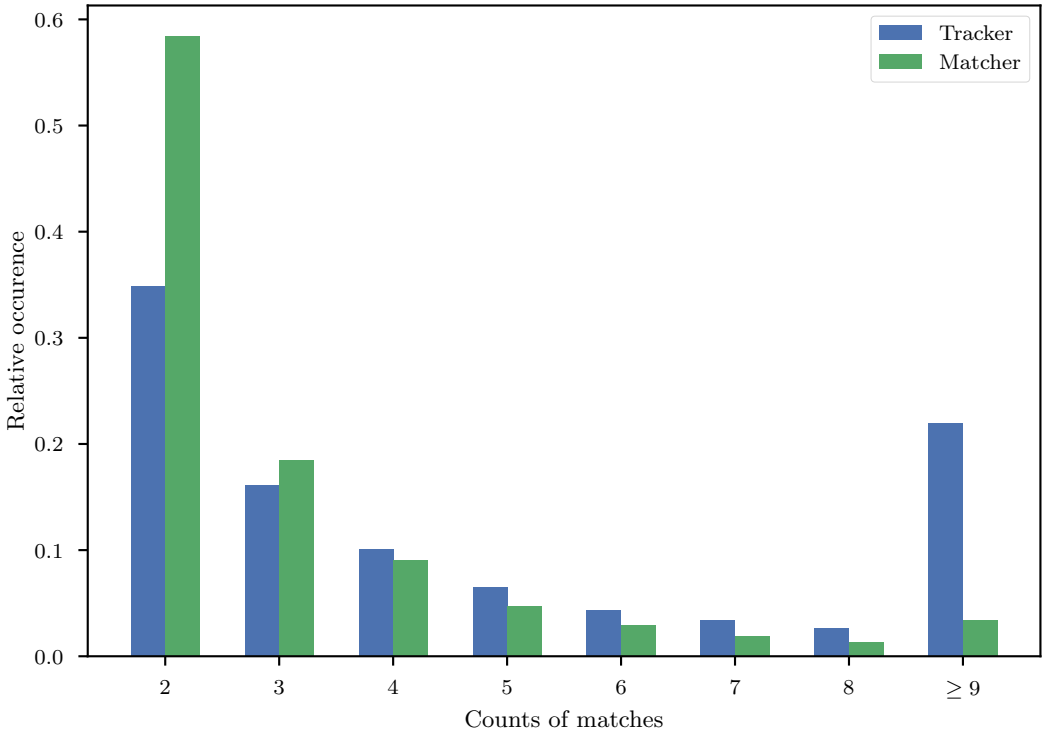


Figure 6.4: Histogram of counts of matches of individual points. The counts are displayed relative to the number of detected points.

A discrepancy can be noticed in the MME metric, where datasets with better MDR score often do not have a better value of the MME metric. We believe that this is again caused by the difference in the number of points and hypothesise that MME is reliable only in the case of a similar number and density of points. Results from the section evaluating the

post-processing step 6.4.4 seem to support this hypothesis.

Crucially for tracking, the number of points visible in the current frame n_k is also significantly better. In figure 6.1 we provide a graph that compares this number over time and shows that the tracking algorithm is almost always more performant, apart from a few drops. We analyse their cause in the following section 6.4.2

The difference between easier and harder datasets can be observed in most of the criteria. However, it is not very profound. A noticeable change in performance in MDR can be observed only between the easy and hard levels. In the dataset Vicon Room 2 medium, most of the metrics even improved compared to the easy level. We might attribute this, however, to the large difference in the trajectory length of the datasets. In the medium datasets, the movement of the drone is more rapid, whereas, in the hard datasets, it is both more rapid and also some abrupt movements with high angular velocity are introduced that often cause motion blur.

We can deduce that higher speeds might not always deteriorate the quality of the solution. This might be caused by the fact that in lower speeds, the pose information gets largely lost in the noise of the propellers. Naturally, too large speeds impact the quality of matching and tracking, and we expect the results to deteriorate under these conditions.

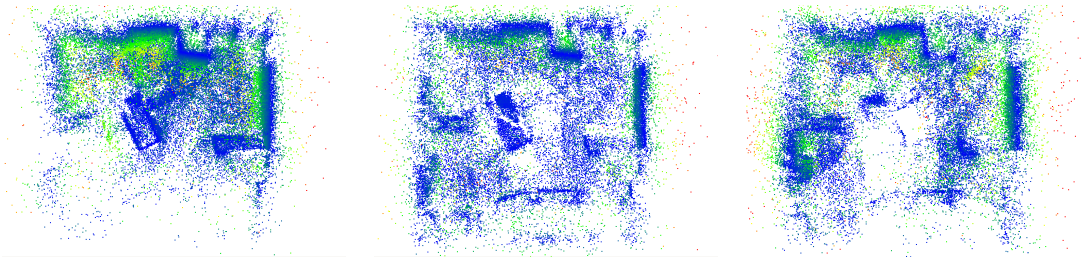
In the figure 6.2 we provide a visual comparison of the resulting point clouds and occupancy grids of both algorithms. The differences in point cloud density and occupancy grid coverage are clearly visible.

Also, in figures 6.3a and 6.3b we provide a visualisation of the MDR of the resulting point clouds. The blue end of the colour spectrum corresponds to points close to the reference, while the red end corresponds to the furthest points.

We can analyse the distribution of the counts of the points' matches, i.e. the number of frames from which they can be observed. This can provide us with some insights into the tracking and matching performance. Figure 6.4 displays the relative histograms of these counts for both algorithms. The points with counts higher than 9 are put into a single histogram box, as after this threshold, the counts continue to decrease, and it is also the value of the triangulation threshold for the tracking algorithm. This means that for the tracking-based algorithm, all the points that are in this category are triangulated. The threshold for the matching-based algorithm is set to 4.

We can see that the matching-based algorithm is more prone to detect only 2 matches of the point and the tracking-based algorithm collects 9 or more matches for a higher number of points. These results are caused by the fact that the matching algorithm is keyframe based while the tracking-based algorithm takes into account the whole information.

In the following sections, we will discuss and evaluate the tracking-based algorithm as we have shown that it significantly outperforms the matching-based algorithm.



(a) V1 easy. (b) V1 medium. (c) V1 hard.

Figure 6.5: Visual comparison of the point clouds produced by the tracking-based algorithm on all the difficulties of the dataset V1 with visualisation of MDR.

6.4.2 Analysis of the tracking algorithm

In this section, we provide a more detailed analysis of some of the inner workings of the tracking-based algorithm. In figures 6.5, one can visually compare the resulting point clouds produced by the tracking-based algorithm on the Vicon room 1 dataset.

Level	**Total** [ms]	**KLT** [ms]	**Detection** [ms]	**Triangulation** [ms]	**Refinement** [s]
Easy	29.5 ± 24.9	15.4 ± 8.6	6.3 ± 2.7	3.15 ± 18.6	2.32 ± 7.48
Medium	24.8 ± 23.7	13.5 ± 7.6	4.6 ± 2.9	2.36 ± 12.4	0.15 ± 0.73
Hard	27.8 ± 19.5	15.9 ± 7.9	5.1 ± 2.8	2.24 ± 9.51	0.30 ± 1.71

Table 6.6: Detailed timing of the tracking-based algorithm on the V1 dataset.

Table 6.6 contains time measurements of different parts of the algorithm on multiple difficulty levels. The total time is composed of the time of the KLT tracker itself, the detection time, the triangulation and graph insertion time, and some other small operations that we do not list here. The duration of the non-linear least-squares refinement should be largely independent of the other durations, as it runs in a separate thread. The refinement loops over points from the last added point until the first and refines points that should be refined. For details about when that is, see section 4.1.6. We measure the time of the refinement loop.

The times do not vary dramatically between difficulty levels. On the refinement time, we can see the result of a higher number of points that have to be optimised in the easiest difficulty level. The same effect can be observed on the triangulation time.

Level	**Ref. iterations**	**Tracked**	**Inliers**	**Delaunay refused**
Easy	3.345 ± 0.729	2020 ± 250	0.990 ± 0.045	0.049 ± 0.048
Medium	3.413 ± 0.571	1698 ± 373	0.967 ± 0.068	0.091 ± 0.057
Hard	3.899 ± 0.965	1742 ± 487	0.948 ± 0.109	0.146 ± 0.103

Table 6.7: Performance statistics of the tracking-based algorithm on the dataset V1.

We further provide some additional statistics of the algorithm in the table 6.7. The values of the average number of the non-linear least-squares refinement iterations per point show that in more complex levels, it is harder to optimise the point's position.

We say that a point is successfully tracked if it has been tracked by the KLT tracker to the subsequent frame and it has passed the verification step by the epipolar geometry. Surprisingly, the average count of tracked points is slightly higher for the hard level than for the medium level. But it can be observed that the standard deviation increases with the difficulty level, which we attribute to the increasing number of erratic movements that tend to disrupt the tracker, as we will show in the graph 6.6.

The number of matches deemed to be inliers decreases with increasing difficulty. This might be the result of the tracker not being able to find the matches as reliably. However, it may also show that the quality of epipolar geometry is worse when a large movement is present.

The percentage of points in the frame that are refused by the Delaunay filtering follows the same trend and could be attributed to the same factors, as well as to less precise Delaunay filtering in the presence of fewer points and more noise.

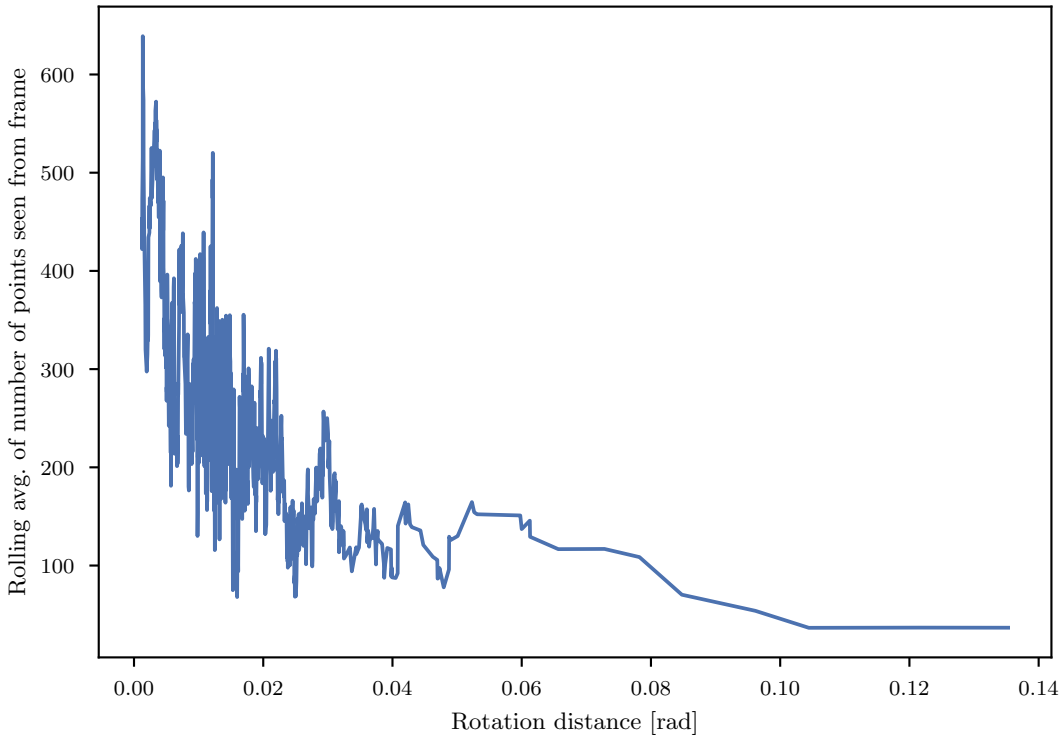


Figure 6.6: Relating the number of visible triangulated points and the rotation distance on the dataset V1 hard. The y-axis is a simple rolling average over a window of 10, to provide a better visualisation of the general trend.

We hypothesise that the before mentioned trends can be attributed to a high rotational distance between frames. We believe that this causes a decrease in the precision and convergence of the KLT tracker. Furthermore, it also seems that the rotation distance itself causes a higher rate of refused points. We believe that this can be only partly attributed to the faults

of KLT tracker and that another cause of this issue is a worse quality of the localisation, from which stems a worse estimate of the epipolar geometry, because in some frames it refuses almost all points, especially when the rotation distance is very large. We ruled out the failure of the tracker to be the cause of these drops by a visual inspection of the matches. The relationship between rotation distance and the number of triangulated points can be seen 6.6. We considered combatting this issue by computing the essential matrix by RANSAC whenever we encountered this major drop in the number of tracked points. This should lead to a more precise estimate of the geometry. However, RANSAC execution produced a drop in framerate that was more damaging than the issue itself.

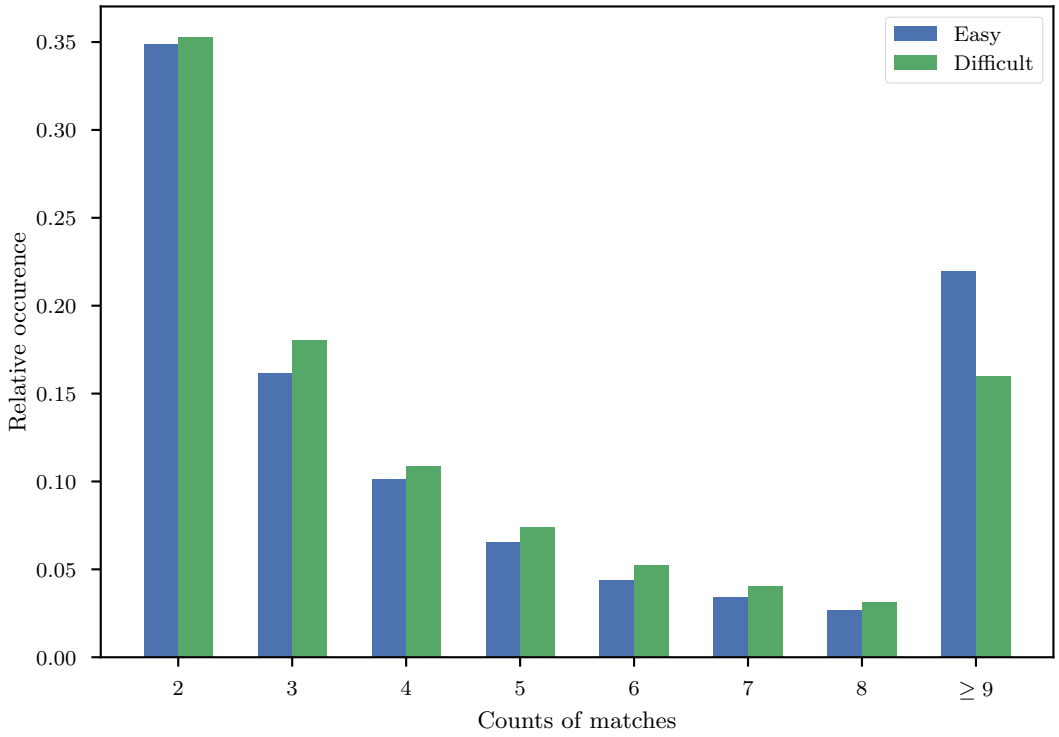
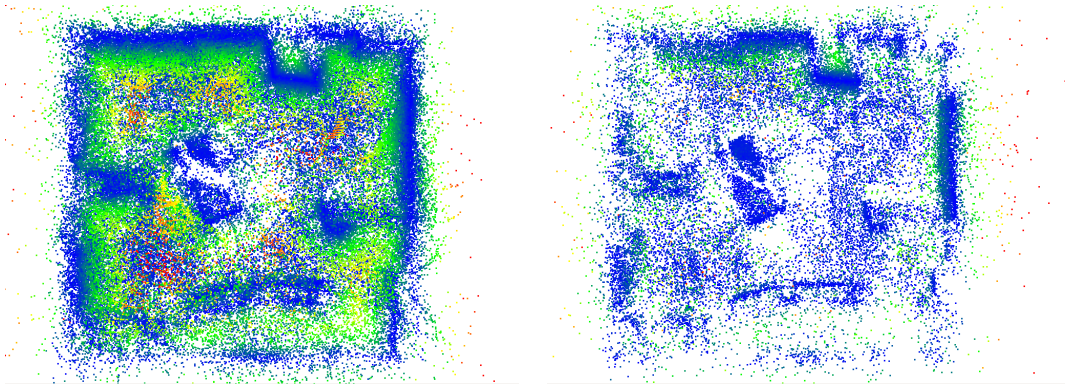


Figure 6.7: Histograms of counts of matches on the datasets V1 easy and difficult

In the figure 6.7 we compare the histogram of the match counts between the datasets V1 easy and V1 difficult. The differences are visible, although not significant, meaning that when the tracker finds a second match for a point, it is almost just as likely to find another match in the hard level as in the easy level.

6.4.3 Evaluation of the Delaunay-based densification

In this section, we evaluate the performance of the Delaunay-based depth filling that we have introduced in section 4.2.3. In addition to the data provided in the table 6.8, we also note that we produced on average around 45 new points per keyframe for the easy level, 115 for the medium level and 25 for the hard level. In the table we also present the results



(a) The Delaunay-only point cloud. (b) The tracking-only point cloud.

Figure 6.8: Visual comparison with MDR visualisation for the dataset V1 medium.

Level	**Alg.**	**MDR** [m]	**MME**	n [$\times 10^3$]	TP	Cov
Easy	orig.	0.149 ± 0.209	-0.447 ± 0.523	≈ 58	0.246	0.129
	combined	0.174 ± 0.223	-0.409 ± 0.517	≈ 80	0.253	0.142
	Delaunay-only	0.241 ± 0.280	-0.263 ± 0.515	≈ 20	—	—
Medium	orig.	0.147 ± 0.222	-0.494 ± 0.713	≈ 35	0.229	0.141
	combined	0.203 ± 0.265	-0.255 ± 0.489	≈ 103	0.206	0.172
	Delaunay-only	0.224 ± 0.272	-0.178 ± 0.432	≈ 68	—	—
Hard	orig.	0.214 ± 0.359	-0.344 ± 0.709	≈ 35	0.325	0.247
	combined	0.264 ± 0.299	-0.287 ± 0.749	≈ 53	0.318	0.260
	Delaunay-only	0.329 ± 0.324	-0.200 ± 0.751	≈ 18	—	—

Table 6.8: Evaluation of the densification on the dataset Vicon Room 1.

if we evaluate the filling point cloud separately without the points produced by the tracking algorithm.

From the results, we can see that the densification is able to significantly increase the number of points. This naturally comes at the price of higher MDR and lower MME, as the linear interpolation often is not a good approximation of the surface and the points defining the triangle might still be erroneous. Furthermore, the densification improves the coverage Cov of the occupation grid. At the same time, it decreases the TP rate, but only to a reasonable degree. We argue that this decrease in the TP rate can be negligible in comparison to the increase in Cov, as we believe that the latter is more valuable for the navigation algorithms.

An interesting result is that the densification adds significantly more points in the medium dataset. This can also be seen in the figure 6.8a. We do not know the cause of this difference in behaviour.

From the visual comparison between the separate filling point cloud in figure 6.8a and the tracking point cloud in figure 6.8b, we can see that the densification successfully enhances the information in some areas, at the cost of introducing additional noise, which aligns with the numerical results from table 6.8.

6.4.4 Evaluation of the post-processing step

We have experimented with multiple versions of the post-processing algorithm outlined in the chapter 5, constructing different combinations of parameters. We list those in the table 6.9.

Short name	BA	Dense	Fixed orient.	SOR	Strict Huber
fixed orient.	✓	✓	✓	✗	✗
fixed orient., strict Huber	✓	✓	✓	✗	✓
fixed orient., SOR	✓	✓	✓	✓	✗
full BA, dense graph	✓	✓	✗	✓	✗
full BA, loop closure	✓	✗	✗	✓	✓
only SOR	✗	—	—	✓	—

Table 6.9: Post-processing algorithms.

We constructed a version with fixed orientation because its estimate should be precise thanks to the use of IMU. In the versions with a strict Huber loss we set its threshold to $\delta = 0.009$, in other versions we set it to $\delta = 0.1$. We do this to limit the effect of outliers and evaluate their influence. We found that for problems of scale similar to ours, we can use the full scene graph in most variants, as it is still computationally tractable and leads to better results. Only in the version with loop closure, we had to select keyframes, because there the optimisation problem became harder with the new connections in the graph, and the computation time has increased dramatically.

In the table 6.10 we present the results of the evaluation of the different variants of the post-processing procedure. Results in bold font are the best within the difficulty level. When multiple results are of similar value, we embolden all of them.

The first important observation from these results is that the algorithms performing the full BA have mostly slightly worse scores in the MDR metric, contrary to what we hoped them to achieve. Even more surprisingly, the MRE is often also worse in these algorithms. Our hypothesis is that these results are caused by the high precision of the VINS algorithm on these datasets. Simply put, the localisation is so precise that the additional BA does not improve it further. On the contrary, the results might deteriorate due to the effect of the outliers. This is the reason why we attempted to construct a version of the post-processing that uses a stricter Huber loss threshold δ. We can indeed see that it outperforms its equivalent with a higher threshold δ, suggesting that our hypothesis might be valid. We believe that in situations where the tracking algorithm produces higher errors, the use of BA in the post-processing will improve the results.

We attribute the deterioration of the MRE to the fact that this is not the actual loss function used in the solution of the BA problem. Rather, as we specified in section 5.1, the loss function is the sum of squared reprojection errors, with the Huber function applied to it. We argue that this also shows the effect of outliers in the full BA.

One of the solutions to this might be to use a strict Huber loss, as we have done, which increases the complexity of the optimisation. This did not help enough; however, an even stricter value of δ maybe would. Another solution is to perform BA with a more aggressively

Level	Post-processing	MDR [m]	MME	MRE [mm]
Easy	no post-processing	0.149 ± 0.209	-0.447 ± 0.523	**3.910 ± 6.102**
	fixed orient.	0.159 ± 0.197	-0.430 ± 0.488	5.247 ± 14.01
	fixed orient., strict Huber	0.158 ± 0.171	-0.575 ± 0.521	4.974 ± 12.60
	fixed orient., SOR	0.131 ± 0.124	-0.441 ± 0.336	—
	full BA, dense graph	0.121 ± 0.121	-0.534 ± 0.315	4.580 ± 9.943
	full BA, loop closure	0.142 ± 0.117	**-0.628 ± 0.327**	30.03 ± 92.37
	only SOR	**0.114 ± 0.124**	-0.571 ± 0.366	—
Medium	no post-processing	0.147 ± 0.222	-0.494 ± 0.713	7.407 ± 9.523
	fixed orient.	0.204 ± 0.264	-0.334 ± 0.744	6.913 ± 8.791
	fixed orient., strict Huber	0.195 ± 0.255	-0.429 ± 0.665	**6.339 ± 8.996**
	fixed orient., SOR	0.126 ± 0.124	-0.440 ± 0.390	—
	full BA, dense graph	0.199 ± 0.230	-0.313 ± 0.407	6.935 ± 8.289
	full BA, loop closure	0.112 ± 0.117	**-0.591 ± 0.453**	29.46 ± 36.65
	only SOR	**0.099 ± 0.106**	**-0.541 ± 0.415**	—
Hard	no post-processing	0.214 ± 0.359	**-0.344 ± 0.709**	**7.274 ± 8.889**
	fixed orient.	0.256 ± 0.366	-0.260 ± 0.683	7.457 ± 8.746
	fixed orient., strict Huber	0.226 ± 0.364	-0.319 ± 0.679	7.542 ± 10.26
	fixed orient., SOR	0.187 ± 0.201	-0.440 ± 0.390	—
	full BA, dense graph	0.201 ± 0.215	-0.280 ± 0.382	8.816 ± 8.935
	full BA, loop closure	0.214 ± 0.212	-0.292 ± 0.380	30.47 ± 34.05
	only SOR	**0.165 ± 0.181**	**-0.339 ± 0.384**	—

Table 6.10: Evaluation of the different versions of post-processing on the dataset V1.

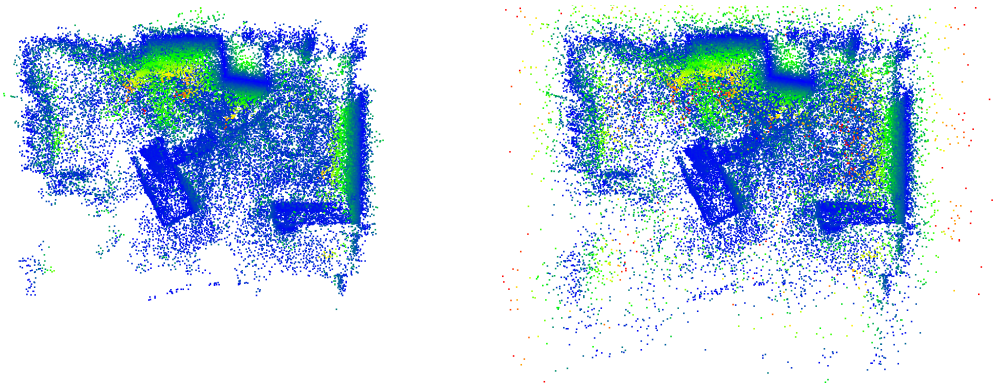
pruned subset of points, which decreases the complexity. However, we would risk removing some important information.

In the version containing the loop closure, we can see that the MRE is significantly higher than in other variants. This is probably caused by the additional matches produced by the loop closure. By inspection of the point clouds produced by this version of the post-processing algorithm, we came to the conclusion that it does remove many outliers, but at the same time, it tends to move some objects in various directions. One of the solutions to this could be to restrain the poses of cameras to some interval around the solution computed by the VINS algorithm.

The simple SOR filter has proven to be the best post-processing tool, especially when the pose information is precise. We note that the number of points removed by the SOR filter is around 1000, which is a low number, considering how much improvement its application yields.

The figures 6.9a and 6.9b visualise the difference between the original point cloud and the same point cloud after the application of the filter. It can be observed, that indeed, many of the gross outliers were removed. This effect is also visible in the graph 6.10, where we serialised the points into a single dimension by their ids and plotted their distance to the nearest reference point. The ids of points are ordered by their occurrence during time.

Another useful piece of information that stems from the data is that the ordering



(a) SOR filter applied. (b) The original.

Figure 6.9: Visual comparison of the effect of SOR filter on the point cloud MDR.

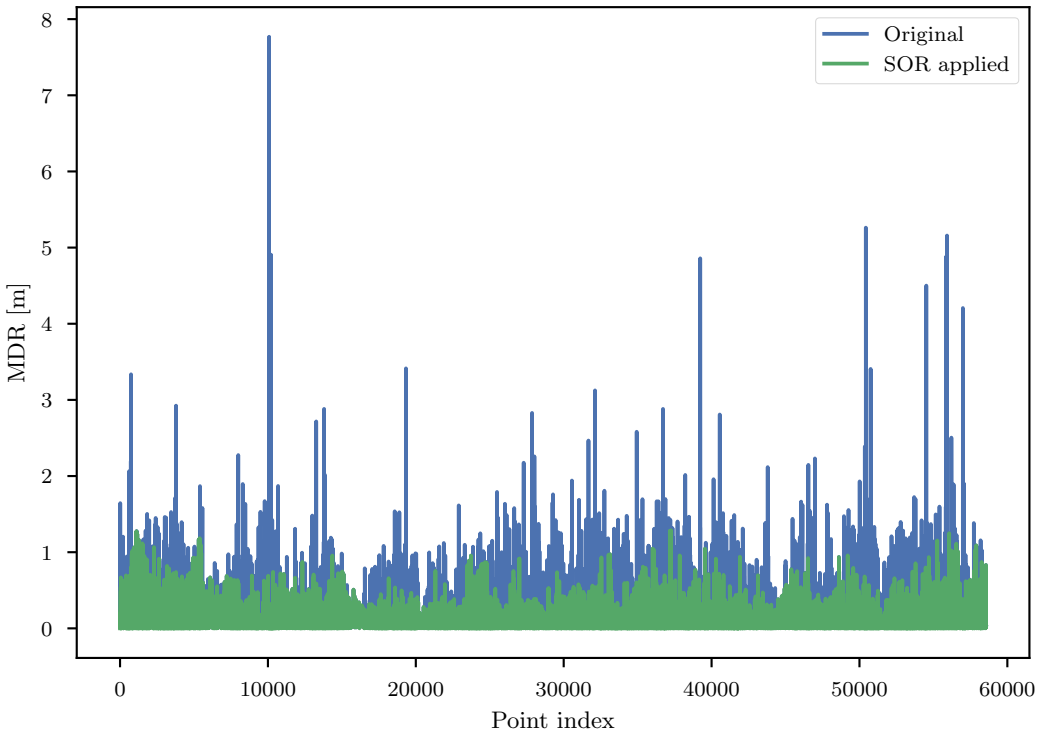


Figure 6.10: Illustrating the effect of applying the SOR filter to the resulting point cloud of the V1 dataset.

provided by MME does roughly correspond to the ordering by the MDR, supporting the conclusions of [67]. This will be useful in the evaluation of the real-world experiments in the section 6.7.

6.4.5 VINS-Mono

Level	Loc.	MDR [m]	MME	MRE [mm]	n [$\times 10^3$]	n_k	TP	Cov
Easy	V-M	0.117 ± 0.184	-0.538 ± 0.527	5.320 ± 7.402	≈ 40	549 ± 194	0.251	0.123
	OV	0.149 ± 0.209	-0.447 ± 0.523	3.910 ± 6.102	≈ 58	578 ± 233	0.246	0.129
Medium	V-M	0.163 ± 0.243	-0.525 ± 0.754	11.36 ± 12.65	≈ 15	199 ± 153	0.195	0.112
	OV	0.147 ± 0.222	-0.494 ± 0.713	7.407 ± 9.523	≈ 35	339 ± 195	0.229	0.141
Hard	V-M	0.210 ± 0.269	-0.408 ± 0.839	10.74 ± 10.62	≈ 16	147 ± 177	0.322	0.174
	OV	0.214 ± 0.359	-0.344 ± 0.709	7.274 ± 8.889	≈ 35	249 ± 203	0.325	0.247

Table 6.11: Comparing VINS-Mono to OpenVINS on the dataset V1.

VINS	V1 easy	V1 medium	V1 hard	V2 easy	V2 medium
OpenVINS	0.642 / 0.076	1.766 / 0.096	2.391 / 0.344	1.164 / 0.121	1.248 / 0.106
VINS-Mono	1.199 / 0.064	3.542 / 0.103	5.934 / 0.202	1.585 / 0.073	2.370 / 0.079

Table 6.12: Values of absolute trajectory error in units degree / metres for the two algorithms, as evaluated in [41].

We demonstrate the flexibility of the developed algorithm by evaluating its performance while using a different underlying localisation algorithm. We pick VINS-Mono [40] for this, as it is used by the MRS group, and we will use it for the following experiments in the simulator and the real-world environment.

The main difference between OpenVINS and VINS-Mono is that VINS-Mono is based on keyframes, that are sampled at a certain frequency. This has an impact on the results in the table 6.11, as the SfM algorithm produces a lower number of points when it uses VINS-Mono. However, this does not have a significant impact on the metrics measuring the quality of the resulting occupancy grid, suggesting that it may be possible to filter keyframes to a certain degree even when using OpenVINS.

In table 6.12 we provide for reference information about the absolute trajectory error of the VINS-Mono and OpenVINS algorithms on the EuRoC MAV datasets as evaluated in [41]. We will not discuss the trajectory evaluation in this work. For the definition of absolute trajectory error and further details on the matter, we refer the reader to **??**. We will only note that it usually is measured in two parts, rotational error and translational error.

6.5 UAV setup

The drone, on which we performed the real-world experiments and also is simulated in the Gazebo environment is a custom-built drone that is based on the DJI F330 frame with Pixhawk 4 flight control unit (FCU).

Importantly for our algorithm, the camera is mvBlueFOX-MLC 200w with fisheye lens and global shutter triggered by the IMU, which ensures precise time calibration of the camera with the IMU. It has a resolution of 752 × 480. The IMU is the IvenSense ICM-20689

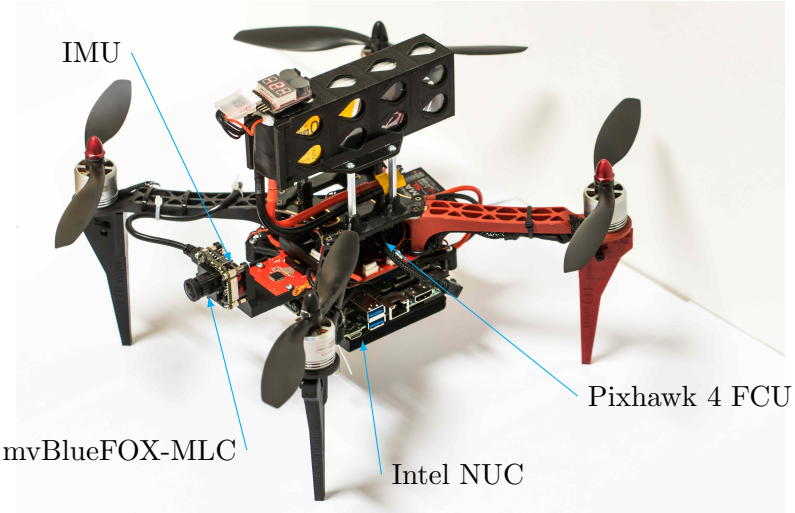


Figure 6.11: The UAV platform.

programmable sensor. The processor on board is the Intel NUC10i7FNK with six cores, base frequency of 1.1 GHz and 16 GB of RAM.

The whole setup can be seen in figure 6.11.

6.6 Gazebo simulation environment

We tested the tracking-based algorithm on a few scenarios in the realistic Gazebo simulator. It simulates the actual drone used by the MRS group and described in section 6.5. The experiments were conducted with VINS-Mono as the underlying localisation algorithm.

The advantage of the simulation is that it provides access to a variety of environments, where we can simulate different flights. We evaluated flights in indoor environments of church and cave, that can be characterised as slow but containing a high number of small rocking movements and vertical movements, which have proven to be challenging for the SfM algorithm.

For the experiments in the church, we evaluated the MDR as the model is in the form of a mesh created from a terrestrial laser-scan. In the figure 6.12, we show the first of the church flights, which is less challenging, as it contains more straight, constant forward motion. The second church flight is more challenging, as it contains a lot of rotating motion, which is difficult for our algorithm.

We did not evaluate it for the cave environment, as it is assembled from smaller models, and it is not so straightforward to obtain the reference point cloud. The cave environment can be seen in the figure 6.13.

6.6.1 Results

In table 6.13, we provide the results of these flights. Indeed, the second church dataset shows worse performance than the first one.

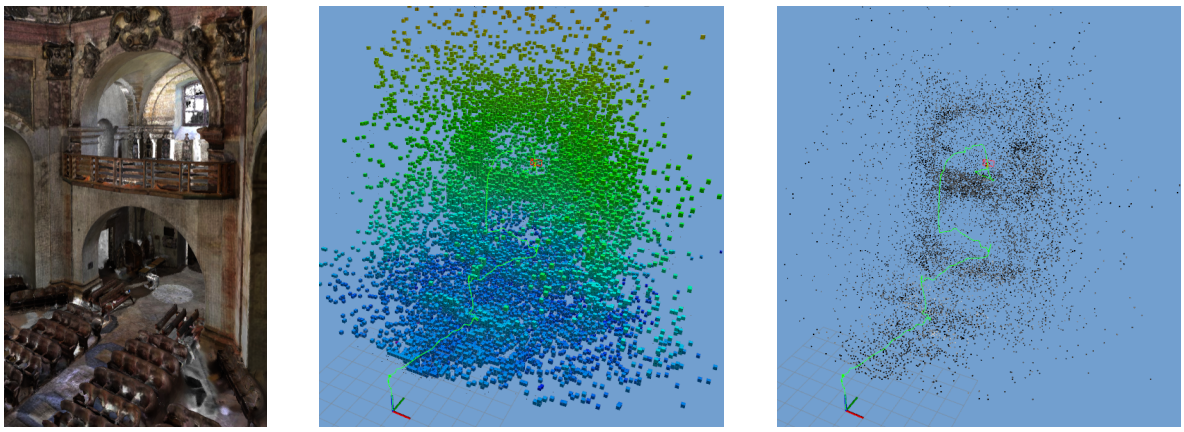


Figure 6.12: Experiment in the Gazebo simulator – the church environment.

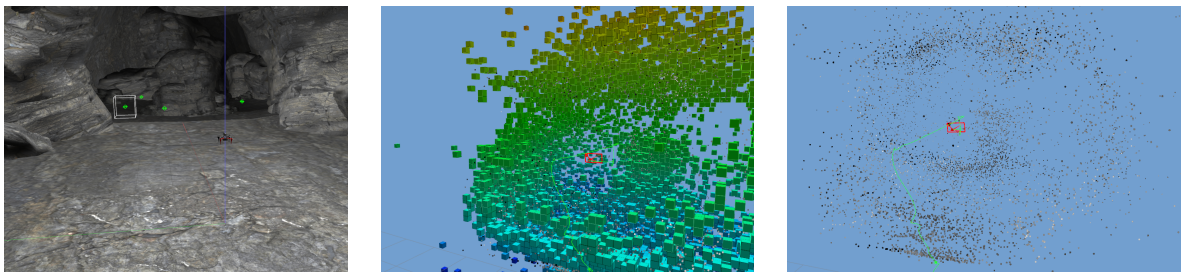


Figure 6.13: Experiment in the Gazebo simulator – the cave environment.

Simulation	**MDR** [m]	**MME**	**MRE** [cm]	n [$\times 10^3$]	n_k
church	0.657 ± 0.700	-1.412 ± 3.078	1.628 ± 3.807	≈ 10	252 ± 166
church, harder	1.087 ± 1.063	-1.166 ± 2.782	1.359 ± 2.715	≈ 16	193 ± 146
cave	—	-0.839 ± 2.234	1.813 ± 7.656	≈ 10	266 ± 205

Table 6.13: Evaluation of performance in the Gazebo simulator.

An important difference between the EuRoC MAV datasets and these simulated environments is the scale of the environments, where the simulated environments have a much larger scale. The challenge posed by this is that two points close in the image become further apart the further they are located from the camera, which decreases the perceived density of the resulting structure. However, as the UAV gets closer to the object, the density should become sufficient.

6.7 Real-world deployment on a UAV

We deployed the algorithm within the MRS UAV system in a test scenario in an outdoor environment of the university courtyard. This poses its challenges compared to the EuRoC dataset, which is largely based in an enclosed indoor environment. One of the main differences

is, similarly to the simulated environments, the scale of the world, which is much greater. Furthermore, the light conditions were more challenging compared to both the simulated environment and EuRoC MAV datasets, when sometimes scenes could get overexposed due to a high amount of natural light and white objects. Also, motion blur occurs in real-world conditions when in the simulation it does not.

Flight	**Duration** [s]	**Description**	**Fig.**
F1	75	the longest flight, straight along the building wall, with the camera pointing in the direction of the building	6.14
F2	62	straight along the building wall, camera pointing in the direction of the motion, where another bulding is located, some trees and varied objects on the left	6.15
F3	51	along a building, trees and a car, with camera pointing in the direction of the objects	6.16
F4	30	UAV flying across the yard, in the direction of a further-away, sunlit, white building, heavy drift	6.17

Table 6.14: Brief description of the real-world flights.

We also provide the .bag data files, which can be used as an alternative to the existing datasets for further testing of localisation and mapping solutions. We provide raw camera and IMU data as well as pose information computed by the VINS-Mono algorithm.

We note that during the experiment, we encountered a problem of the drift of the VINS-Mono pose estimates. We believe that this is due to the high noise produced by the rotating propellers. Therefore, the quality of the localisation deteriorates over time, which has the effect of substantially changing the scale of the reconstruction. Because of this, we decided to shorten the flights from their original length for our evaluation. In table 6.14 we provide some information about the flights.

We also increased the parameter of the Delaunay threshold to 1.5, as the outdoor environments include more objects that violate the assumption of the local planarity of the environment, such as trees. Due to the bigger scale, we also increased the maximum optimisation movement to 10 metres.

6.7.1 Results

Flight	**Duration** [s]	**MME**	**MRE** [cm]	n [$\times 10^3$]	n_k
F1	75	-0.857 ± 1.566	0.6799 ± 1.277	≈ 20	772 ± 178
F2	62	-1.904 ± 2.412	1.323 ± 2.599	≈ 11	538 ± 165
F3	51	-1.069 ± 1.858	1.022 ± 1.394	≈ 8,8	602 ± 209
F4	30	-1.724 ± 2.642	0.8707 ± 1.589	≈ 2,5	363 ± 162

Table 6.15: Evaluation of performance in the real-world conditions.

In table 6.15 we provide the results of the evaluation of the system on these flights. The key metric for the navigation algorithms, the average number of triangulated points visible

from the frame n_k, is comparable and even greater than in the EuRoC MAV datasets. The MRE metric is higher, which is expected due to the greater scale of the environment.

In figures 6.14 – 6.17 we show for individual flights the footage of the drone flying in the environment, what it sees, including the current tracked points and the Delaunay graph between them, and the path and the resulting OctoMap of the environment.

In figures 6.18 – 6.22, we show the progress of the algorithm on Flight 1 in the first 50 seconds, with frames separated by 10 seconds. In these figures, we can see from left to right: the view from the drone, including the Delaunay triangulation, the resulting point cloud, the occupancy grid.

We believe that the results of these real-world experiments demonstrate that the proposed algorithm performs well under those circumstances.

Figure 6.14: Flight 1.

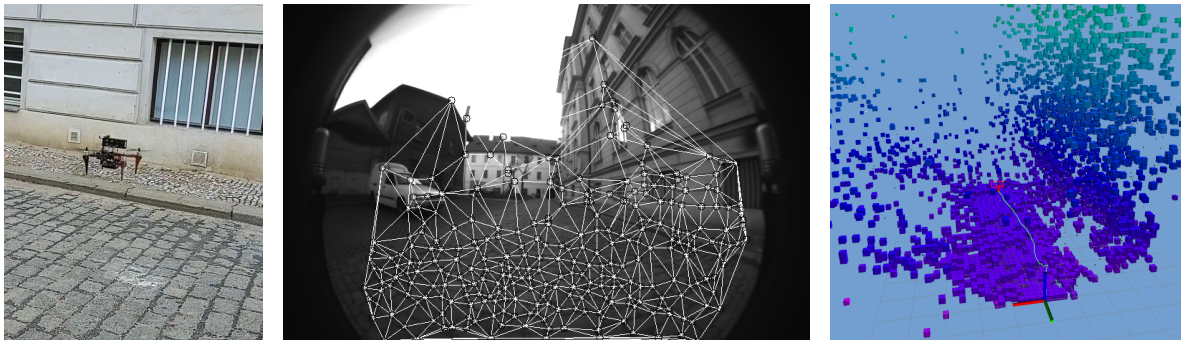


Figure 6.15: Flight 2.

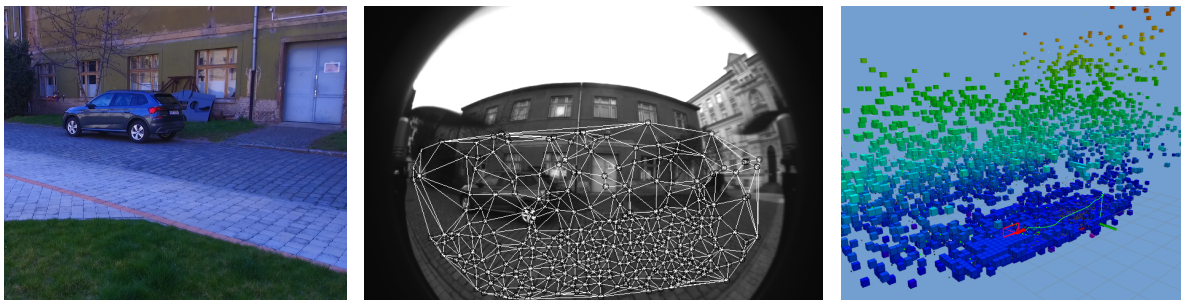


Figure 6.16: Flight 3.

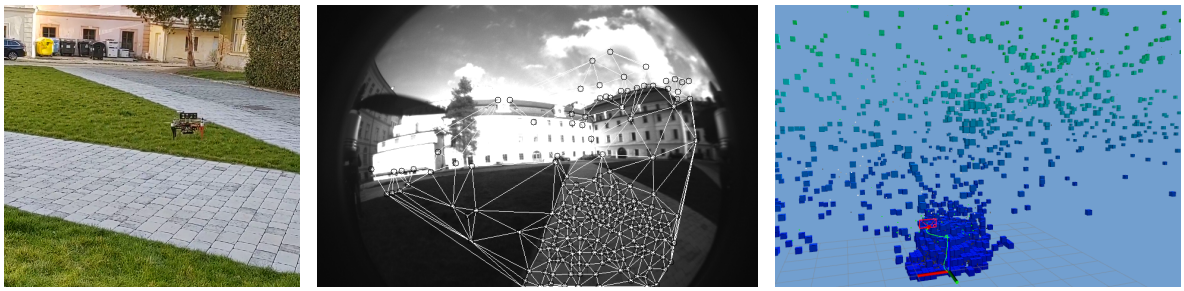


Figure 6.17: Flight 4.

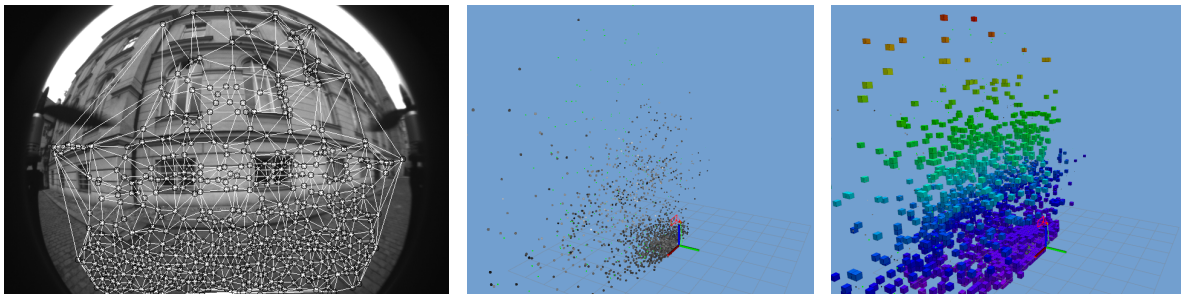


Figure 6.18: 10 seconds into the flight 1.

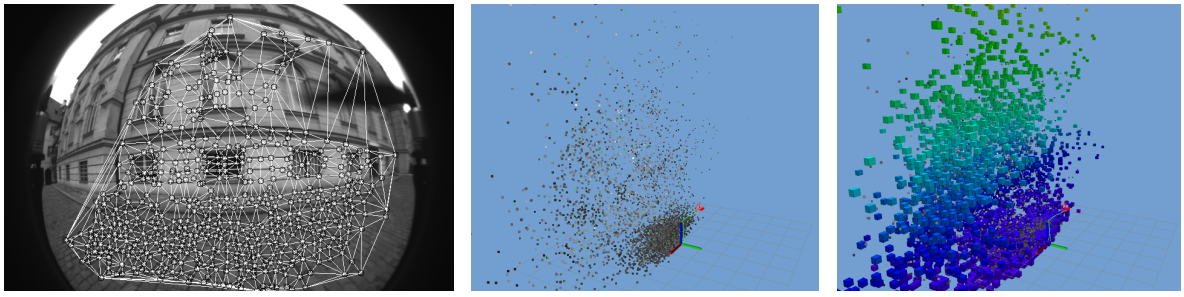


Figure 6.19: 20 seconds into the flight 1.

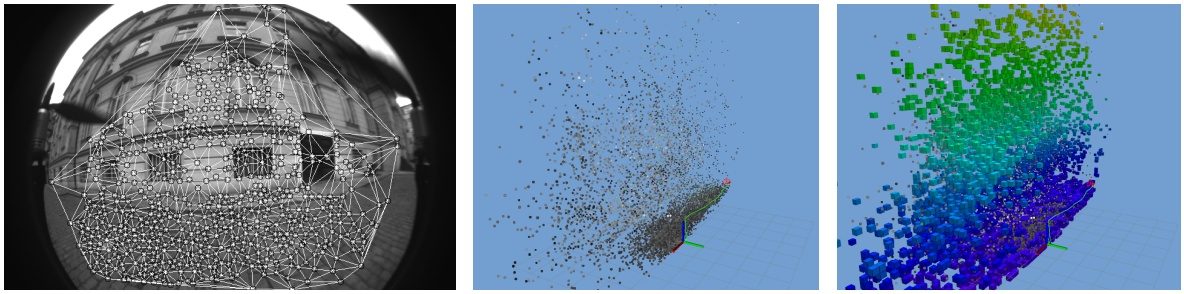


Figure 6.20: 30 seconds into the flight 1.

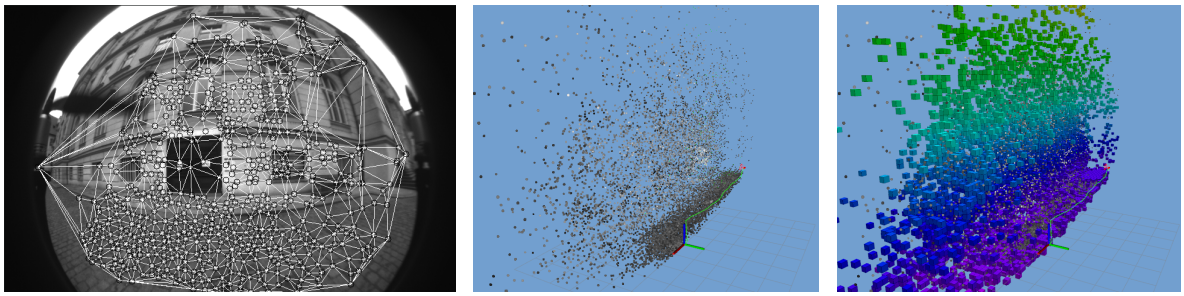


Figure 6.21: 40 seconds into the flight 1.

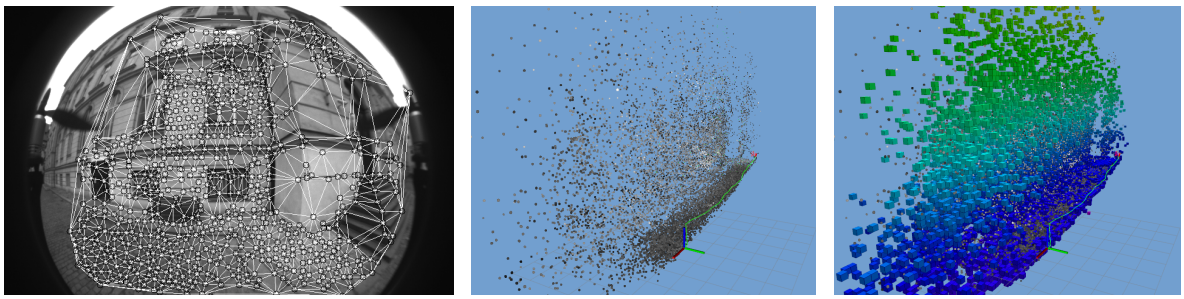


Figure 6.22: 50 seconds into the flight 1.

Chapter 7

Conclusion

In this thesis, we have first motivated and outlined the problem of real-time monocular 3D scene reconstruction for autonomous navigation of a UAV equipped with an IMU. We have conducted a detailed survey of the state-of-the-art literature about 3D reconstruction, focusing on the SfM and SLAM problems and showed how they relate to our problem. We have then laid out the fundamentals of multiple-view computer vision and non-linear least-squares optimisation, providing the theoretical background necessary to solve the problem at hand.

We have devised and implemented two algorithms for this problem. Both of them are intended as an extension to an existing localisation technique, more specifically VINS. The first algorithm is based on the matching approach with keyframes, while the second one is built around a KLT tracker and does not require subselection of keyframes. For the second, tracking-based algorithm, we created methods for outlier filtering and point cloud and depth filling based on the 2D Delaunay triangulation of a set of points visible from a single frame. We have described these algorithms in detail, which we then followed by an extensive evaluation of their performance on the EuRoC MAV datasets. Furthermore, they were integrated into the MRS system and verified in a simulated environment before testing them on an actual UAV in real-world conditions. As an interface between our algorithm and path-planning algorithms, we have chosen the probabilistic occupancy grid framework OctoMap. Lastly, we have presented possible post-processing steps that can be utilised to improve the quality of the structure obtained during flight and we have evaluated their performance.

In summary of the evaluation results, we found that the tracking-based algorithm performs better than the matching-based algorithm in almost all of the tested criteria. Also, we have found that a large rotational distance between frames is problematic for the tracking-based algorithm. We have found that the Delaunay point cloud filling can significantly densify the resulting point cloud, leading to better performance scores in the coverage of the occupancy grid. The main result of the evaluation of the post-processing steps was that BA with loop closure does not improve the quality of the structure, which we believe is due to the already high precision of the localisation, and we discussed this in more detail. We have found the SOR outlier filter to be the most useful post-processing tool, as it can remove most of the outlier points.
In this work, we have fulfilled all of the assigned tasks, namely:

1. We designed and implemented an algorithm for 3D scene reconstruction from monocular camera images. Pose estimates from a self-localisation system of the UAV were used to initialise the camera poses.
2. We verified the developed scene reconstruction technique on publicly available EuRoC dataset, which contains indoor flight sequences of a UAV equipped with cameras
3. We integrated the developed method into the Multi-robot Systems Group UAV system
4. We conducted experiments in the realistic Gazebo simulator to verify the integrated system
5. The feasibility of deployment onto the actual hardware platform flying in a real-world environment was demonstrated, and we also discussed results and parameter selection.
6. The reconstruction process was extended by a post-processing step, which can be run offline after the flight to improve the accuracy of the reconstructed structure.
7. We evaluated the improvement in quality of the post-processed structure compared to the structure obtained during the flight.

7.1 Future work

Although we have selected the algorithms' parameters empirically, we acknowledge that there is space for a more detailed analysis of the interplay between the individual parameters. Also, the map produced by the algorithm remains to be verified for collision-free motion planning in an autonomous-flight scenario. Furthermore, for mapping over longer periods of time, some mechanism of saving old data to the memory should be implemented.

One way to improve the tracking-based algorithm could be to alter the formulation of the KLT tracker to take into account the information about the known epipolar geometry. Furthermore, direct methods could be potentially used for tracking. The depths on the Delaunay graphs could be smoothed, possibly providing better results. We believe that a promising direction of enhancing the density of the structure would be to improve the state-of-the-art depth filling deep neural networks to be able to run in real time on a CPU. Another research direction is to develop a version of these algorithms for GPU-enabled setups. In that case, variational approaches for a fully dense 3D reconstruction can be utilised, possibly enhanced by some deep neural network method. Then, the resulting post-processed 3D structure could be used to complete the scans produced by a terrestrial 3D scanner. Research direction opposite to the previous one is to adapt these algorithms for use in an even more resource-restricted application in smaller devices.

Bibliography

[1] T. Rouček, M. Pecka, P. Čížek, T. Petříček, J. Bayer, V. Šalanský, D. Heřt, M. Petrlík, T. Báča, V. Spurný, F. Pomerleau, V. Kubelka, J. Faigl, K. Zimmermann, M. Saska, T. Svoboda, and T. Krajník, *DARPA Subterranean Challenge: Multi-robotic Exploration of Underground Environments*, 2019.

[2] V. Spurny, V. Pritzl, V. Walter, M. Petrlik, T. Baca, P. Stepan, D. Zaitlik, and M. Saska, "Autonomous Firefighting Inside Buildings by an Unmanned Aerial Vehicle," *IEEE Access*, vol. 9, pp. 15 872–15 890, 2021.

[3] P. Ješke, Š. Klouček, and M. Saska, "Autonomous Compact Monitoring of Large Areas Using Micro Aerial Vehicles with Limited Sensory Information and Computational Resources," in *Lecture Notes in Computer Science*, vol. 11472 LNCS, 2019.

[4] G. Silano, J. Bednar, T. Nascimento, J. Capitan, M. Saska, and A. Ollero, "A Multi-Layer Software Architecture for Aerial Cognitive Multi-Robot Systems in Power Line Inspection Tasks," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021.

[5] P. Petráček, V. Krátký, and M. Saska, "Dronument: System for Reliable Deployment of Micro Aerial Vehicles in Dark Areas of Large Historical Monuments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2078–2085, 2020.

[6] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[7] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles," 2020.

[8] H. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," in *Readings in Computer Vision*. Elsevier, jan 1987, pp. 61–62.

[9] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer, "A survey of structure from motion," *Acta Numerica*, vol. 26, pp. 305–364, jan 2017.

[10] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[11] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building Rome in a day," in *Proceedings of the IEEE International Conference on Computer Vision*, 2009, pp. 72–79.

[12] J. Heinly, J. L. Schönberger, E. Dunn, and J.-M. Frahm, "Reconstructing the World* in Six Days *(As Captured by the Yahoo 100 Million Image Dataset)," *IEEE Conference on Computer Vision and Pattern Recognition*, no. January, pp. 3287–3295, 2015.

[13] K. Häming and G. Peters, "The structure-from-motion reconstruction pipeline - A survey with focus on short image sequences," *Kybernetika*, vol. 46, no. 5, pp. 926–937, 2010.

[14] J. L. Schonberger and J.-M. Frahm, "Structure-from-Motion Revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-Decem. IEEE, jun 2016, pp. 4104–4113.

[15] C. Wu, "Towards linear-time incremental structure from motion," in *Proceedings - 2013 International Conference on 3D Vision, 3DV 2013*, 2013, pp. 127–134.

[16] D. Crandall, A. Owens, N. Snavely, and D. Huttenlocher, "Discrete-continuous optimization for large-scale structure from motion," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2011, pp. 3001–3008.

[17] R. Gherardi, M. Farenzena, and A. Fusiello, "Improving the efficiency of hierarchical structure-and-motion," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07/80, no. 2, 2010, pp. 1594–1600.

[18] X. F. Han, H. Laga, and M. Bennamoun, "Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1578–1604, 2021.

[19] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, nov 2004.

[20] H.-J. Chien, C.-C. Chuang, C.-Y. Chen, and R. Klette, "When to use what feature? SIFT, SURF, ORB, or A-KAZE features for monocular visual odometry," *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pp. 1–6, 2016.

[21] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 2011.

[22] D. Detone, T. Malisiewicz, and A. Rabinovich, "SuperPoint: Self-supervised interest point detection and description," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2018-June, 2018, pp. 337–349.

[23] M. Muja and D. G. Lowe, "Fast matching of binary features," in *Proceedings of the 2012 9th Conference on Computer and Robot Vision, CRV 2012*, 2012, pp. 404–410.

[24] M. A. Fischler and R. C. Bolles, "Random sample consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, jun 1981.

[25] R. Raguram, O. Chum, M. Pollefeys, J. Matas, and J. M. Frahm, "USAC: A universal framework for random sample consensus," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 2022–2038, 2013.

[26] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, feb 2009.

[27] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment – a modern synthesis," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1883, 2000, pp. 298–372.

[28] C. Stachniss, J. J. Leonard, and S. Thrun, "Simultaneous localization and mapping," in *Springer Handbook of Robotics*, 2016, pp. 1153–1175.

[29] D. Chekhlov, A. P. Gee, A. Calway, and W. Mayol-Cuevas, "Ninja on a Plane: Automatic Discovery of Physical Planes for Augmented Reality Using Visual SLAM," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE, nov 2007, pp. 153–156.

[30] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, "The SLAM problem: A survey," in *Frontiers in Artificial Intelligence and Applications*, vol. 184, no. 1, 2008, pp. 363–371.

[31] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," pp. 1–11, jun 2017.

[32] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007, pp. 225–234.

[33] R. Mur-Artal, J. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, oct 2015.

[34] D. Cremers, "Direct methods for 3D reconstruction and visual SLAM," in *Proceedings of the 15th IAPR International Conference on Machine Vision Applications, MVA 2017*, 2017, pp. 34–38.

[35] J. Stühmer, S. Gumhold, and D. Cremers, "Real-time dense geometry from a handheld camera," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6376 LNCS, 2010, pp. 11–20.

[36] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct monocular SLAM," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8690 LNCS, no. PART 2, 2014, pp. 834–849.

[37] J. Engel, V. Koltun, and D. Cremers, "Direct Sparse Odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018.

[38] W. N. Greene and N. Roy, "FLaME: Fast Lightweight Mesh Estimation Using Variational Smoothing on Delaunay Graphs," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, 2017, pp. 4696–4704.

[39] G. Huang, "Visual-inertial navigation: A concise review," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., jun 2019, pp. 9572–9582.

[40] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[41] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "OpenVINS: A Research Platform for Visual-Inertial Estimation," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2020, pp. 4666–4672.

[42] F. Ma and S. Karaman, "Sparse-to-Dense: Depth Prediction from Sparse Depth Samples and a Single Image," in *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., sep 2018, pp. 4796–4803.

[43] A. Wong, X. Fei, S. Tsuei, and S. Soatto, "Unsupervised depth completion from visual inertial odometry," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1899–1906, apr 2020.

[44] S. Y. Loo, A. J. Amiri, S. Mashohor, S. H. Tang, and H. Zhang, "CNN-SVO: Improving the mapping in semi-direct visual odometry using single-image depth prediction," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., may 2019, pp. 5218–5223.

[45] F. Aleotti, G. Zaccaroni, L. Bartolomei, M. Poggi, F. Tosi, and S. Mattoccia, "Real-time single image depth perception in the wild with handheld devices," *Sensors (Switzerland)*, vol. 21, no. 1, pp. 1–17, jan 2021.

[46] D. Wofk, F. Ma, T. J. Yang, S. Karaman, and V. Sze, "FastDepth: Fast Monocular Depth Estimation on Embedded Systems," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2019, pp. 6101–6108.

[47] J. Ku, A. Harakeh, and S. L. Waslander, "In defense of classical image processing: Fast depth completion on the CPU," in *Proceedings - 2018 15th Conference on Computer and Robot Vision, CRV 2018*. Institute of Electrical and Electronics Engineers Inc., dec 2018, pp. 16–22.

[48] F. Ma, L. Carlone, U. Ayaz, and S. Karaman, "Sparse depth sensing for resource-constrained robots," *International Journal of Robotics Research*, vol. 38, no. 8, pp. 935–980, jul 2019.

[49] D. Brown, "Decentering Distortion of Lenses," *Photometric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[50] J. Kannala and S. S. Brandt, "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 8, pp. 1335–1340, 2006.

[51] D. Q. Huynh, "Metrics for 3D rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, 2009.

[52] H. Cheng and K. C. Gupta, "An historical note on finite rotations," *Journal of Applied Mechanics, Transactions ASME*, vol. 56, no. 1, pp. 139–145, 1989.

[53] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed. Springer, New York, NY, 2006.

[54] S. Agarwal, K. Mierle, and Others, "Ceres solver," .

[55] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. II, 2005, pp. 1508–1515.

[56] ——, "Machine learning for high-speed corner detection," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3951 LNCS. Springer Verlag, 2006, pp. 430–443.

[57] B. Lucas and T. Kanade, "Iterative Technique of Image Registration and Its Application to Stereo," in *Proceedings of the International Joint Conference on Neural Networks*, 1981.

[58] C. Tomasi, C. Tomasi, and T. Kanade, "Detection and Tracking of Point Features," *INTERNATIONAL JOURNAL OF COMPUTER VISION*, vol. 9, pp. 137—-154, 1991.

[59] J. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker," *Intel Corporation*, vol. 1, no. 2, pp. 1–9, 2001.

[60] S. Baker and I. Matthews, "Lucas-Kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, 2004.

[61] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[62] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011.

[63] M. Quigley and Others, "ROS: an open-source Robot Operating System," in *IEEE ICRA workshop on open source software*, 2009, pp. 1–5.

[64] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[65] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," , 2010.

[66] Z. Zhang and D. Scaramuzza, "A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry," in *IEEE International Conference on Intelligent Robots and Systems*, 2018, pp. 7244–7251.

[67] J. Razlaw, D. Droeschel, D. Holz, and S. Behnke, "Evaluation of registration methods for sparse 3D laser scans," in *2015 European Conference on Mobile Robots, ECMR 2015 - Proceedings*, 2015.

Appendices

Archive Content

In table 1 are listed names of all root directories and files in the archive accompanying the thesis. The archive and other accompanying material can be found on the webpage[1].

Name	Description
thesis	the thesis in pdf format
vins_sfm	the code of the algorithms
README.txt	additional information about the code

Table 1: CD Content

[1] `http://mrs.felk.cvut.cz/melecky2021thesis`

Acronyms

AR augmented reality. 8

BA bundle adjustment. 7, 37, 38, 46, 55, 56, 65

BRIEF Binary Robust Independent Elementary Features. 26

CPU central processing unit. 3, 9–11, 35, 66

EKF extended Kalman filter. 8, 10

FAST Features from Accelerated Segment Test. 25, 26, 30, 42, 43

FCU flight control unit. 58

GPS Global Positioning System. 3, 7

GPU graphics processing unit. 5, 9–11, 66

IMU inertial measurement unit. v, 3, 9, 10, 55, 58, 65

KLT Kanade–Lucas–Tomasi. viii, 30, 31, 42, 43, 51–53, 65, 66

LIDAR light detection and ranging. 8, 11, 35, 36, 41, 44, 45

MAV Micro Aerial Vehicle. viii, 41, 44, 45, 47, 49, 51, 53, 55, 57, 58, 60–62, 65

MDR mean distance to reference. 42

MME mean map entropy. 44

MRE mean reprojection error. 44

MRS Multi-robot Systems. 2, 4, 10, 35, 60

ORB Oriented FAST and Rotated BRIEF. 6, 8, 24–27, 38, 39, 42

PCL Point CLoud Library. 39, 42

PnP Perspective-n-Point. 7

RAM random-access memory. 59

RANSAC random sample consensus. 7, 19, 53

ROS Robot Operating System. 41

SfM structure from motion. 3–8, 10, 11, 13, 18, 19, 58, 59, 65

SIFT scale-invariant feature transform. 6, 24, 38, 39

SLAM simultaneous localization and mapping. vii, 3, 4, 7–11, 45, 65

SOR statistical outlier removal. 39, 42, 56, 57, 65

UAV unmanned aerial vehicle. v, xi, 1–3, 9, 17, 44, 46, 48, 60, 61, 65

VINS visual-inertial navigation system. v, 9, 10, 23, 24, 35, 45, 46, 56, 65