DISS. ETH NO. 25546

Planning for Autonomous Micro-Aerial Vehicles with Applications to Filming and 3D Modeling

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH Zürich

(Dr. sc. ETH Zürich)

presented by

Benjamin Hepp

Diplom-Physiker
Ruprecht-Kalrs-Universität-Heidelberg

born 20.05.1986
citizen of Germany

accepted on the recommendation of

Prof. Dr. Otmar Hilliges

Prof. Dr. Marc Pollefeys
Dr. Neel Joshi

2018

Abstract

Camera equipped micro aerial vehicles (MAVs), and in particular multi-copters, have become affordable and abundant in recent years both in the professional and in the consumer sector. They are used in areas as diverse as recreation, filming, structural inspection and monitoring for agriculture. Their success can be attributed to their relatively simple mechanical and electronic design, their flexibility in terms of possible motion and their smooth dynamics which are easy to handle from a control perspective.

In most of these use cases the MAVs are still deployed in a manual fashion where one or more human experts steer the MAV or perform related work. This can be partly attributed to the difficulty of autonomous flight in GPS-denied environments. However, even with a good GPS signal the design of autonomous systems requires the incorporation of domain knowledge from the corresponding use case. This is very challenging as this knowledge of experts is often intuitive and non-formal. In this thesis we demonstrate the design of such systems that take sensory information and provide a sequence of decisions to solve the given task. We will identify and formalize the goals for solving the task and the constraints that need to be fulfilled and incorporate both into a planning scheme that will provide us with a sequence of decisions to achieve the goals.

In the first part of the thesis we will focus on a framework that allows us to generate flight trajectories that demonstrate smooth and

pleasant motion of the camera. The constraints come in the form of the MAV dynamics and the goals are derived from a keyframe based description of the desired film composition. This leads to a constrained optimization problem over the MAV trajectory that we can solve with non-linear programming.

The second part of the thesis introduces a system that provides flight paths to capture an image collection that is suitable for reconstructing high quality 3D models of a user-defined region of interest. Here the major constraints are battery time of the MAV and collision free motion. We formulate this task as a submodular orienteering problem on a graph of possible viewpoints and provide an approximate solver.

The third and final part of the thesis looks at the autonomous exploration of unknown environments. Here, the goal is to discover as much surface in as little time as possible. To achieve this goal we define an expert policy with full knowledge of the environment and train a 3D convolutional neural network to imitate this expert policy. The resulting model can then be used to explore an unknown environment.

Zusammenfassung

In den letzten Jahren konnte ein Preisrückgang und eine entsprechende Zunahme von Drohnen mit Kameras beobachtet werden. Hierbei handelt es sich insbesondere um Multikopter die sowohl im professionellen als auch im privaten Sektor zum Einsatz kommen. Die Verwendung umfasst eine weite Bandbreite von Unterhaltung und Filmen über Strukturüberwachung im Baugewerbe bis hin zu Beobachtung und Kontrolle in der Landwirtschaft. Der Erfolg solcher Drohnen kann auf mehrere Faktoren zurückgeführt werden: ihren einfachen mechanischen und elektronischen Aufbau; die Erreichbarkeit beliebiger Orte; ihre kontinuierliche Dynamik die eine einfache Regelung und Steuerung erlaubt.

In den meisten dieser Anwendungsfälle erfolgt der Einsatz der Drohnen manuell durch ein oder mehrere Personen welche die Steuerung und unterstützende Funktionen übernehmen. Die manuelle Steuerung kann teilweise damit erklärt werden, dass der autonome Flug von Drohnen in Umgebungen ohne GPS Empfang mit hohen Schwierigkeiten verbunden ist. Doch auch wenn ein gutes GPS Signal vorhanden ist muss beim Entwurf von autonomen Systemen entsprechendes Fachwissen bezüglich des Einsatzbereiches und der Zielsetzung einfliessen. Da dieses Fachwissen oft auf viel Erfahrung basiert ist der Entwurf solcher Systeme sehr schwierig. In dieser Arbeit demonstrieren wir den Entwurf solch autonomer Systeme: aufgrund von Sensor-Daten trifft das System eine Abfolge von Entscheidungen um die Zielset-

zung zu erreichen. Für eine gegebenen Anwendungsfall werden wir die Zielsetzung und nötige Einschränkungen identifizieren. Diese werden dann in einer Planungs-Methode integriert welche die erforderlichen Entscheidungsschritte zum Erreichen der Zielsetzung berechnet.

Der erste Teil der Arbeit befasst sich mit einem System welches die Erzeugung von Flugtrajektorien mit kontinuierlicher und angenehmer Kamerabewegung ermöglicht. Die Einschränkungen sind hier durch die Dynamik der Drohne gegeben und die Zielsetzung leitet sich aus einer Beschreibung der Filmkomposition an einigen Schlüsselstellen ab. Diese Formulierung führt zu einem Optimierungsproblem mit Zwangsbedingungen über die Flugtrajektorie und kann mit Nichtlinearer Programmierung gelöst werden.

Im zweiten Teil der Arbeit beschäftigen wir uns mit der Aufnahme von Bildern welche die Rekonstruktion von qualitativ hochwertigen 3D Modellen ermöglicht. Zu diesem führen wir ein System ein welches Flugtrajektorieren erzeugt während denen geeignete Bilder aufgenommen werden um einen Bereich von Interesse zu rekonstruieren. Einschränkungen sind vor allem durch die Batterielaufzeit der Drohne und kollisionsfreie Flugpfade gegeben. Wir formulieren und lösen die Aufgabenstellung als ein sogenanntes *submodular orienteering* Problem welches auf einem Graph möglicher Kamerakonfigurationen definiert ist.

Im dritten und letzten Teil der Arbeit betrachten wir die autonome Erkundung unbekannter Gebiete. Hier ist das Ziel möglichst viel Oberfläche in möglichst kurzer Zeit zu entdecken. Um diese Zielsetzung zu erreichen definieren wir eine Expertenstrategie welche alle Informationen über das unbekannte Gebiet kennt. Wir trainieren dann ein künstliches neuronales Netzwerk um die Expertenstrategie nachzuahmen. Das neuronale Netzwerk kann dann verwendet werden um ein unbekanntes Gebiet zu erkunden.

Acknowledgement

First I would like to thank my advisor, Otmar Hilliges, for his advise and guidance in the academic process and for giving me the chance to do a Ph.D. in his group despite time constraints on my side. In his group I had a lot of freedom to choose what I was working on which allowed this thesis to span several topics of my own interest. Thanks also to Marc Pollefeys and Neel Joshi for joining my Ph.D. committee.

Thanks to Fabrizio for our joint suffering on the Deformables project. I am grateful that I had two great office mates, Christoph and Jie. Thanks to Tobi for endless discussions on Manifolds and other topics. I really enjoyed the gaming sessions and pizza evenings with Christoph, Jie and Emre. Thanks to Christoph, Fabrizio, Stephan and Tobi for our joint work. And of course thanks to Adrian, Manuel, Velko, Wookie and all other AIT members.

Furthermore I want to thank all collaborators, Matthias Nießner, Neel Joshi, Sudipta N. Sinha, Dey Debadeepta, Ashish Kapoor, Moritz Baecher and Bernhard Thomaszewski. I also appreciate all the help and discussion with other members of AIT and ETH in general.

I also want to say thanks for all the administrative support from ETH.

I am grateful for Caroline and her constant support during my time in the AIT group. Last but not least I want to thank my parents and siblings for their support and especially my parents for feeding me oatmeal when I was a kid. I am sure it helped.

Contents

Chapter 1

Introduction

In recent year micro aerial vehicles (MAVs) have become affordable and abundant both in the professional and in the consumer sector. We can nowadays find MAVs in numerous applications: racing as a recreational activity, hobby and high-end filming applications, structural inspection work on construction sites and historic buildings or transport machinery like trains and planes, inspection and exploration of hard to reach or hazardous environments, security surveillance of large properties, inspection and monitoring for agriculture but also for plant- and wild-life in protected areas and 3D modeling of landscapes, buildings or heritage sites. The majority of these MAVs are in the form of multi-copters and their success and widespread adoption can be attributed to their relatively simple mechanical and electronic design, their smooth dynamics that are easy to handle from a control perspective and the flexibility in terms of space that they can reach.

Despite this proliferation of MAVs most of their use cases are still performed in a manual way, i.e. there is one or multiple human experts steering the MAV or performing other support work. To some extent this can be attributed to the fact that robust autonomous flight in GPS-denied environments (i.e. indoor or crowded urban areas with

bad signal quality) is still an open research problem and no off-the-shelf solutions exist. However, another challenging aspect is the design of autonomous systems that incorporate the necessary domain knowledge which is often intuitive and non-formal.

In this thesis we try to bridge this gap between usage of MAVs and their autonomous deployment. We design systems that plan trajectories or sequences of decisions for MAVs with applications in the areas of filming, 3D modeling and exploration and we put them in context to related work from trajectory optimization, sensor placement, active vision and sequential decision making.

1.1 Problem Statement

The schematic in Fig. 1.1 depicts the parts of a robotic system used for a specific task. There are basically two ways to solve a given task in this setting: 1) Use a human expert with domain knowledge to control the robot. Here the human expert plans and performs the necessary steps to reach the goal. 2) Design a system that plans and autonomously performs the necessary steps to reach the goal. Here the system designer incorporates required domain knowledge into the system.

While a large part of perception and control, in particular lower levels, can be formulated and developed in a task independent manner there is usually a significant portion that is specific or at least needs to be adapted to the task at hand. As such a designer for a planning system has to tackle and solve problems both in the area of perception and control but also in translating the often intuitive domain knowledge into an algorithmic approach.

In this thesis we tackle the problem of designing such planning systems that can incorporate sensory information and provide a sequence of decisions to solve the task at hand. We demonstrate this design process for tasks in the area of filming, 3D modeling and exploration. In each case we will identify goals that we want to achieve and constraints that we need to fulfill. We will then incorporate these goals

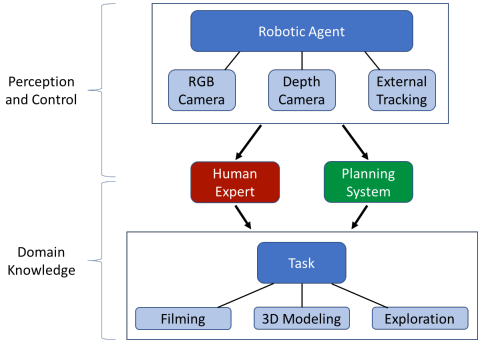

Figure 1.1: The schematic depicts the parts of a robotic system used for a specific task. In this thesis we focus on designing planning systems for autonomous robots by showing examples in filming and 3D modeling.

and constraints into a planning scheme that will generate a trajectory or sequence of decisions leading to the goal.

In the case of filming the goals are a smooth and pleasing motion of the camera and aesthetic aspects of the composition that we try to put in a formal setting. The constraints come in the form of the MAV dynamics, the coarse composition of the scene and the subject to be filmed and obstacles that need to be avoided. We will formulate the planning problem as a constrained optimization problem over the trajectory of the MAV discretized in time.

For the 3D modeling task the goal is to reconstruct a high-quality model of the scene. This will results in the task of selecting a sequence of camera viewpoints that will be used for the Structure from Motion (SfM) and Multiview-Stereo (MVS) processing and can be traversed within the maximum flight time of the MAV. The set of resulting images should fulfill certain criteria so that our resulting 3D model is accurate and complete. We formulate this as maximizing the information captured by the set of images. When this notion of

information is modeled so that the contributed information of different measurements is independent of each other this information score often exhibits a property called *submodularity* [1]. The related sensor placement problem where the aim is to maximize a coverage function has been studied intensely in the literature for the submodular setting [1]. Our task is different in that we try to find a trajectory of sensor positions that fulfills a travel budget while maximizing a submodular function and is known as *submodular orienteering*.

The exploration task is somewhat related to the 3D modeling task but in contrast we want to choose the next image viewpoint without a prior model of the scene. Here, the 3D model is updated on the fly and parts of the scene will be completely unknown. A good notion of information should then capture the expected distribution of surface in the scene so that measurements are taken that discover more surface and not free space. Typically such systems create an occupancy map that is updated with each new measurement. After each update the system chooses from a set of new measurement positions the one with the maximum expected information gain. This information gain measure is usually hand-crafted and based on a raycast into the occupancy map. However, it is difficult to incorporate prior knowledge about a scene into such a measure and as a result different information gain measures work well on different types of scenes. Instead we formulate the computation of the information gain function in a data-driven fashion by making use of an expert policy during training that has access to ground truth models of the scene.

1.2 Design Choices

In the previous section we discussed the problems and task that we are investigating in this thesis. Here we discuss possible options and explain our choices regarding hardware platform, planning algorithms and data representations.

Hardware Typical examples of robotic platforms are wheel-driven ground robots which can carry heavy loads and can provide long running times on battery but their movement range is limited to the ground and even small steps can be an insurmountable obstacle. Thus a ground based robot is strongly constrained in the space that it can reach which is of high importance for tasks such as filming or large scale 3D modeling where a high flexibility in the camera placement is desired. As an example, a wheel-driven robot is unable to acquire images from parts of a building's surface that are not visible from the ground or that are too far away or have very steep viewing angles from those ground positions. This will often prevent the computation of distance values for these surfaces. Another platform are autonomous planes. These can carry a downward looking camera and provide image data for large areas in a short time. However, these planes usually have to fly at altitudes of $100m$ to prevent collisions as their dynamics strongly constrain maneuverability and an obstacle recognition system would have to be very fast. This makes them unsuitable to provide close-up footage of surface details that are not observable from a birds-eye perspective. Their dynamics also make them unsuitable for tasks where a position should be kept or only changed slowly such as in filming. A third option is that of a multi-copter, i.e. a drone capable of vertical take off and landing and motion along 4 degrees of freedom (DOFs) (i.e. 3 position DOFs and 1 rotation DOFs). A gimbal mounted camera can provide full 6 DOF for the camera placement. This allows for arbitrary camera motion that is only temporally constrained by the dynamics of the system making it suitable to filming as well as taking images from above as well as from the side of a building enabling the creation of 3D models of a whole building including parts that are not observable from the ground and parts that are not observable from a plane at high altitudes. This flexibility comes at the price of a lower range and battery time compared to a plane and less lifting capability than a ground robot but for the tasks at hand we deem the advantages to clearly outweigh the disadvantages.

Possible sensors include normal color or grayscale cameras, stereo cameras, depth cameras or laser range scanners (also known as LI-

DARs). Clearly, for filming we desire a high-quality color camera but for other tasks such as 3D modeling or exploration other sensors are also suitable and we briefly discuss them here. Laser range scanners offer very high accuracy and allow direct distance measurements both indoors and outdoors. Unfortunately, their cost and weight make them a unsuitable for many applications, especially in an airborne setting. Depth cameras are a cheaper alternative and also provide direct distance measurements through projected light or via time-of-flight measurements. However, they suffer from degraded performance in outdoor scenarios with direct sunlight and provide restricted ranges of distance measurements. Stereo cameras do not suffer from degradation in outdoor scenarios but are less accurate due to ambiguity when establishing dense correspondences between the image pair and similar to depth cameras provide only a limited distance range. In contrast to these sensors color or grayscale cameras can not provide direct distance measurements. However, similar to a stereo camera, multiple images from different positions can provide information about the structure of a scene. Often the process is split into two steps: First the camera position and orientations are computed based on sparse correspondences of salient feature points, this process is called *Structure from Motion* (SfM). In the second step, known as *Multi-View Stereo* (MVS) dense correspondences are established between the known cameras and used to triangulate the depth of individual pixels Because of their flexibility, low weight and affordable price we chose an RGB camera as the primary sensor.

Planning Algorithms The choice of robotic platform (and sensor) imposes constraints on the flight trajectory in terms of the dynamics and the travel budget due to limited battery life. These constraints have to be accounted for when planning for a sequence of decisions, i.e. selecting the set of positions at which the camera should take an image. At the same time the camera motion should achieve a goal, i.e. the set of selected image positions should maximize some notion of information about the scene, so that we make the best use of the avail-

able budget. We distinguish two different formulations of the goal. In tasks such as filming the continuous camera motion is of interest for the goal (i.e. smooth footage and composition of each frame) whereas in other tasks such as 3D modeling and exploration only a sparse set of camera positions is of interest for the task (a good set of image viewpoints). In the first case we model the planning as a continuous optimization problem. In the second case we model the planning as a discrete optimization problem and we will see connections to the fields of sensor placement and orienteering.

Data Representation When it comes to the 3D modeling and exploration task the most common, explicit representations for 3D models are triangular meshes and occupancy maps. Triangular meshes are collections of triangles that make up the surface of an object. They are the de facto standard in graphics applications as they can represent arbitrary shapes and levels of detail. In robotic applications however, triangular meshes have the disadvantage that they only record information about the surface of a scene but they do not posses any information about occupied, free or unknown space. While occupied space can be inferred from closed and waterproof meshes it is often difficult to acquire such meshes in a robust manner. In any case a distinction between free and unknown space is not possible. A representation that allows this distinction is that of an occupancy map. Here the space is typically divided into a regular grid of cube-shaped voxels with a fixed size. Each voxel can then carry information about its occupancy (i.e. how much of its volume is occupied) and about its certainty (i.e. how certain or probable is its occupancy). While an occupancy map can be stored memory-efficient in a hierarchical fashion in the form of an octree the memory requirements are still approximately cubic to the inverse cube length of a voxel. Thus a triangular mesh and an occupancy map offer different advantages and disadvantages and in this thesis we are interested in both of them. Renderings of both representations for the same scene are shown in Fig. 1.2. Fundamentally, both representations can be created from

Figure 1.2: Shown on the left is a color image of a church. The middle pane shows a rendering of a mesh representation and the right pane shows a rendering of the occupancy map representation.

depth images (i.e. images where each pixel value represents the distance from camera projection center to the object visible in the pixel) so in both cases we need to acquire a set of depth images from our scene.

1.3 Challenges and Contributions

In this thesis we develop components for developing autonomous systems aimed at quadrotors. These components include a local trajectory generation framework, an occlusion-aware system to compute viewpoint sequences for 3D modelling and a novel, data-driven approach to compute functions based on local views of incomplete occupancy maps. Here we describe these components and corresponding contributions in more detail.

Optimization-Based Planning of Quadrotor Trajectories The generation of trajectories for quadrotors and other MAVs is an interesting and timely problem as the control of an MAV equipped with a gimbal-

controlled camera is non-trivial even for experienced users. Furthermore, it is not a priori clear what constitutes a good trajectory in different application scenarios. If we take filming as an example, a novice user might want to just film a small number of interesting points in sequence with little regard for timing while an expert user could specify a timing that the MAV should closely follow while giving freedom to the path the MAV follows. To encompass different requirements a flexible framework is necessary that incorporates a model of the MAV to ensure that the generated trajectories can be followed with high fidelity. To address these difficulties we propose an optimization-based framework that leverages non-convex optimization to generate feasible trajectories for quadrotors. The framework is able to incorporate user-provided high-level goals such as visually pleasing video shots, optimal racing trajectories or aesthetically pleasing motion (see Chapter 3).

Trajectory Planning for Multi-View Stereo Reconstruction Stepping away from filming another interesting use case for camera equipped MAVs is the creation of 3D models due to their flexibility and relatively low hardware and operating costs. In particular the generation of detailed 3D models for crowded urban scenarios is challenging even with the combinations of camera-equipped cars and planes due to non-observable regions. We conjecture that with the emerging success of virtual and augmented reality headsets (AR/VR) and the increase of autonomous MAVs and autonomous vehicles there will be a high demand for capturing high-quality 3D models of our surroundings. This will facilitate the blending of the virtual and real world for AR/VR applications and the safe operation of autonomous robots. However, recording image sets with high overlap and coverage that facilitate high-quality 3D reconstructions is challenging even for expert users. In this setting we contribute an optimization strategy for autonomous urban 3D modeling with MAVs that incorporates travel budget and free-space constraints and maximizes the observed surface in a volumetric, occlusion-aware representation of the scene. This is enabled

by the formulation of a *submodular* coverage function for a set of images and a novel algorithm for solving the resulting *submodular orienteering* problem. In this context we also provide a quantitative comparison with other methods (see Chapter 4).

Learning a viewpoint utility score Finally, a related problem is the autonomous exploration of unknown environments. Compared to the setting described above we do not have a strong prior knowledge about the area that we want to map such as in mapping of caves or catastrophic sites or indoor environments. Autonomous mapping and exploration in this setting is particularly challenging as we need to account for safe motion of the robot and also want to map the environment efficiently in terms of required time. At the same time we want to incorporate vague prior knowledge about typical 3D structures. To address these challenges we propose to learn a utility function for scoring viewpoints in a partly observed volumetric map. Instead of hand-crafting heuristics of our prior knowledge we devise a multi-scale representation of the map that can be fed into a 3D convolutional neural network and train it to match the output of an oracle with access to the ground truth map The learned utility function is then used to descide on the next viewpoint during exploration (see Chapter 5).

To summarize, in this thesis we describe several case studies of designing planning systems for different vision-related tasks. The challenges arise from the need to translate task-specific domain knowledge into identifiable goals and constraints for an autonomous system. We believe that this work provides a detailed discourse on these tasks and the corresponding transfer of domain knowledge and also serves as a reference for the development of autonomous systems for related tasks. In addition we report promising directions for future work that we discuss in Chapter 6.

1.4 Thesis Outline

The structure of this thesis is as follows:

Chapter 2 gives an overview of techniques and methods that provide the foundations for the following chapters. The topics covered range from 3D modeling over trajectory planning to neural networks. Due to the large span of topics the description is kept concise and references for further reading are given.

Chapter 3 introduces an algorithm for generating flight trajectories that follow a sparse set of 3D locations provided by a user. These 3D locations can be augmented with desired viewpoints that a gimbal-controlled camera should film. Additionally, we include the possibility for the user to define obstacles and certain aesthetic properties of the film composition. We formulate a continuous, non-linear optimization problem that we solve to compute the desired trajectories.

Chapter 4 describes a system for generating flight trajectories to efficiently scan 3D models of building-scale structures in urban, crowded environments. These flight plans can be readily executed on an MAV equipped with a gimbal-controlled camera and the resulting images can be used to perform Structure-from-Motion and Multi-View Stereo to acquire a digital 3D model of the scanned scene.

Chapter 5 approaches the problem of exploring an unknown environment with a robotic camera. To this end an expert policy is defined by using ground-truth information of the scene. The expert policy is used to generate training data that is used to learn a utility function that predicts the usefulness of a new view. The utility function is modeled as a 3D convolutional neural network and based on a multi-scale representation of a 3D occupancy map.

Chapter 6 concludes the thesis and puts it in context with open questions and interesting directions for future work.

The core contributions of the thesis are contained in Chapter 3, 4 and 5. Because of the different application domains presented in each chapter we present a separate section covering related work in each chapter.

1.5 Publications

The core contributions of this thesis are based on the following publications:

- Christoph Gebhardt*, Benjamin Hepp*, Tobias Nägeli, Stefan Stevsic, Otmar Hilliges.
  Airways: Optimization-Based Planning of Quadrotor Trajectories according to High-Level User Goals.
  Conference on Human Factors in Computing Systems (CHI), 2016.

- Benjamin Hepp, Debadeepta Dey, Sudipta N. Sinha, Ashish Kapoor, Neel Joshi, Otmar Hilliges.
  Learn-to-Score: Efficient 3D Scene Exploration by Predicting View Utility.
  European Conf. on Computer Vision (ECCV), 2018.

- Benjamin Hepp, Matthias Niessner, Otmar Hilliges.
  Plan3D: Viewpoint and Trajectory Optimization for Aerial Multi-View Stereo Reconstruction.
  Accepted for publication in *Transactions on Graphics (TOG), ACM*, 2018.

Further publications that were conducted during the course of my PhD research but are out of scope of this thesis are listed below:

*These authors contributed equally to the work

- Benjamin Hepp*, Tobias Nägeli*, Otmar Hilliges.
  Omni-directional person tracking on a flying robot using occlusion-robust ultra-wideband signals.
  Int. Conf. on Intelligent Robots and Systems (IROS), 2016.

- Moritz Bächer, Benjamin Hepp*, Fabrizio Pece*, Paul G. Kry, Bernd Bickel, Bernhard Thomaszewski, Otmar Hilliges.
  DefSense: Computational Design of Customized Deformable Input Devices.
  Conference on Human Factors in Computing Systems (CHI), 2016.

- Ankit Gupta*, Benjamin Hepp*, Mustafa Khammash.
  Noise induces the population entrainment of incohernt uncoupled intracellular oscillators.
  Cell Systems, 3(6), 2016.

- Benjamin Hepp, Ankit Gupta, Mustafa Khammash.
  Adaptive Hybrid Simulations for Multiscale Stochastic Reaction Networks.
  The Journal of Chemical Physics, 142(3), 2015.

*These authors contributed equally to the work

Chapter 2

Background

In this chapter we will cover basics that are necessary to understand the following chapters and will point the reader to additional references for more details. We start by introducing methods for trajectory planning that will be used later on. Next we explain the concept of occupancy maps and their construction and representation as an octree. The following section will give an overview of submodular optimization and orienteering and relevant work in that field. After this we will give an overview of 3D modeling including camera modeling, Structure from Motion and Multi-View Stereo. Finally, we finish with a quick overview on 3D convolutional neural networks and how to train them.

2.1 Trajectory Planning

Here we look at the problem of finding a trajectory or a path from a start configuration (i.e. the position and orientation of a robot) to a goal configuration or a set of goal configurations. We assume that we are given a map denoting free and occupied (or unknown) space so that we can determine if a path would lead to a collision.

We denote the configuration space as $\mathcal{C}$ and a metric measuring the distance or cost of going from one configuration to another as $Cost : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$. Let $Collision : \mathcal{C} \to 0, 1$ be a function that tells us if a specific configuration would lead to a collision (1) or not (0). We overload this function with a second variant $Collision : \mathcal{C} \times \mathcal{C} \to 0, 1$ that tells us if a path from one configuration to another one would lead to a collision. Furthermore, we assume that we have a model *Step* of the robot that returns the next configuration given an input $u \in \mathcal{U}$ where $\mathcal{U}$ is the input space:

$$Step : \mathcal{C} \times \mathcal{U} \to \mathcal{C} \quad . \tag{2.1}$$

Note that we assume a fixed time discretization of the model. The task of finding a path from a start configuration c_s to a goal configuration g_s is then formulated as

$$\min_{p \in \mathcal{P}} \sum_{i=1}^{i=|\mathcal{P}|} Cost(P_{i-1}, P_i) \tag{2.2}$$

$$\text{such that } \begin{cases} \mathcal{P}_0 = c_s \\ \mathcal{P}_{|\mathcal{P}|} = c_g \\ Collision(\mathcal{P}_{i-1}, \mathcal{P}_i) = 0 \ \forall i \in 0, \ldots, |\mathcal{P}| \\ \exists u \in \mathcal{U} : Step(\mathcal{P}_{i-1}, u) = \mathcal{P}_i \ \forall i \in 1, \ldots, |\mathcal{P}| \end{cases} , \tag{2.3}$$

where $\mathcal{P}$ is the set of all finite sequences of $\mathcal{C}$. In general this problem is very hard to solve as the configuration space can be of high dimension and the function *Collision* is often highly non-convex. We point out here that the underlying formulation leading to (2.2) is often a discretization of the continuous robot dynamics by direct collocation [2, 3].

Nonlinear Programming In certain situations the problem (2.2) can be solved by nonlinear programming solvers. In the general case these solvers will require an initial trajectory (which is often required to fulfill the constraints) and will only find a local minimum around this

initial path. This local minimum can be far worse than the global minimum depending on the initial path and the constraints and finding a *good* initial path is a difficult problem itself (if we know how to find a *good* initial path we are basically reducing the problem to a local minimization problem). In chapter 3 we use this approach to compute trajectories for a quadrotor that are designed to be visually pleasing and observe objects of interest in a scene. As described here we discretize the continuous dynamics and solve the resulting nonlinear program by Sequential Quadratic Programming (SQP).

Mixed-Integer Programming Another approach is to formulate the problem as a mixed-integer program by splitting the space into convex regions and introducing categorical variables that indicate to which convex region each piece of the trajectory belongs. The problem can then be solved by an off-the-shelf solver for mixed-integer semidefinite programs [4]. While these solvers can be very efficient in finding a global optimum the mixed-integer problem is NP-complete and thus there is no guarantee that a solution will be found. However, one should note that after an initial feasible solution has been found one can simply stop the optimization after a computational or time budget has been expended and go forward with the best solution found so far. We also point out here that the problem itself is slightly relaxed as the set of convex regions often does not cover the full space.

Randomized Planning In chapter (4) we need to plan trajectories through an urban crowded scene. As mentioned earlier a nonlinear programming solver can only find a locally optimal solution and especially when many obstacles are present such a solution can be considerably worse than the optimal one. Instead we resort to sampling based planning to solve Eq. (2.2). In particular we use the RRT$*$ method [5, 6, 7]. The core idea is to incrementally build a tree of configurations that will eventually connect the start configuration (root of the tree) with the goal configuration. To this end we need an additional function $Steer : \mathcal{C} \times \mathcal{C} \to \mathcal{U}$ which takes configurations $c_1 \in \mathcal{C}$

and $c_2 \in \mathcal{C}$ and returns an input $u \in \mathcal{U}$ that moves the robot from c_1 towards c_2. This can be formulated as an optimization problem:

$$\mathit{Steer}(c_1, c_2) = \operatorname{argmin}_{u \in \mathcal{U}} \mathit{Cost}(\mathit{Step}(c_1, u), c_2) \quad , \tag{2.4}$$

but for simpler models we can invert the derivative of the model dynamics *Step*, linearize it at c_1 and solve for the input $u \in \mathcal{U}$ that moves us in the direction of $c_2 - c_1$.

We first explain the simpler RRT algorithm and then extend it to RRT$*$. RRT works as follows: Initially, the tree T consists only of the root (the start configuration). In each iteration we sample a random configuration $c_{\text{rand}} \in \mathcal{C}$ from our configuration space such that $\mathit{Collision}(c_{\text{rand}}) = 0$ (this sampling is often biased towards the goal configuration). We search for the configuration

$$c_{\text{nearest}} = \mathit{argmin}_{c \in T} \mathit{Cost}(c, c_{\text{rand}}) \tag{2.5}$$

that is closest to c_{rand} according to *Cost*. Next we compute a new configuration that is moving towards c_{rand}:

$$u_{\text{next}} = \mathit{Steer}(c_{\text{nearest}}, c_{\text{rand}}) \tag{2.6}$$

$$c_{\text{next}} = \mathit{Step}(c_{\text{nearest}}, u_{\text{next}}) \quad . \tag{2.7}$$

If $\mathit{Collision}(c_{\text{nearest}}, c_{\text{next}}) = 0$ we add c_{next} to the tree T with c_{nearest} as its parent node. In [8] the authors show that RRT provides *probabilistic completeness* meaning that given enough iterations the probability of finding a solution, which fulfills certain regularity conditions, converges to 1.

However, in [6, 7] the authors demonstrate that RRT does not provide an optimality guarantee and introduce the variant RRT$*$ that provides a *asymptotic optimality* under suitable conditions, i.e. given enough iterations the cost of the found solution converges to the global optimum. RRT$*$ improves upon RRT in two ways: 1) if there exists a node c_{min} in the neighborhood of c_{next} with a smaller distance from it than c_{nearest} then c_{min} becomes the parent of c_{next}; 2) every node c in the neighborhood of c_{next} which can be reached from c_{next} with a

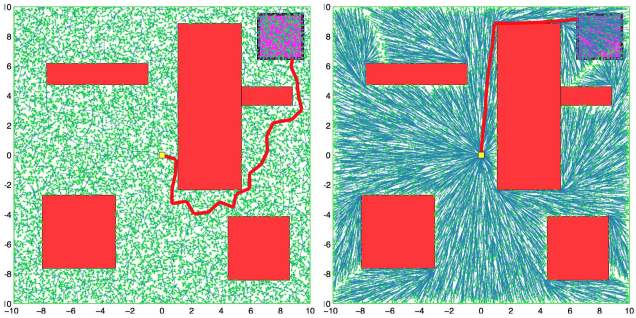

Figure 2.1: Shown is a figure from [7] that shows a tree built by RRT on the left side and a tree built by RRT∗ on the right side. The yellow square is the start configuration, the pink square the set of goal configurations, the red rectangles are obstacles and the red line represents the best found path. Clearly, RRT∗ found a more optimal solution than RRT.

lower accumulated cost than so far is reconnected to c_{next}, i.e. its new parent becomes c_{next}. The algorithm is given in Alg. 1. An example of a built tree and the best found path for RRT and RRT∗ is shown in Fig. 2.1.

Algorithm *RRT∗*:
- **Input:** Start configuration c_{start}
- **Input:** Goal configuration c_{goal}
- **Input:** Number of iterations N
- **Output:** Tree T
- $T \leftarrow c_{\text{start}}$; **for** $i \leftarrow 0$ *to* N **do**
  - $c_{\text{rand}} \leftarrow$ Sample from $\mathcal{C}$;
  - $c_{\text{nearest}} \leftarrow$ Nearest node in T;
  - $u_{\text{next}} \leftarrow$ *Steer*$(c_{\text{nearest}}, c_{\text{rand}})$;
  - $c_{\text{next}} \leftarrow$ *Step*$(c_{\text{nearest}}, u_{\text{next}})$;
  - **if** *Collision*$(c_{nearest}, c_{next}) = 0$ **then**
    - Continue to next iteration;
  - **end**
  - // Check for a better connection;
  - $c_{\text{min}} \leftarrow c_{\text{nearest}}$;
  - *NearNodes* $\leftarrow$ Nodes in T from neighborhood of c_{new};
  - **for** $c \in$ *NearNodes* **do**
    - **if** *Collision*$(c, c_{next}) = 0$ *and* *AccCost*(c) + *Cost*$(c, c_{next}) <$ *AccCost*(c_{min}) + *Cost*(c_{min}, c_{next}) **then**
      - $c_{\text{min}} \leftarrow c$;
    - **end**
  - **end**
  - Add edge $c_{\text{min}} \rightarrow c_{\text{next}}$ to T;
  - // Rewiring of the tree;
  - **for** $c \in$ *NearNodes* **do**
    - **if** *Collision*$(c_{next}, c) = 0$ *and* *AccCost*(c_{next}) + *Cost*$(c_{next}, c) <$ *AccCost*(c_{next}) **then**
      - Set c_{next} as the new parent of node c;
    - **end**
  - **end**
- **end**

End

Algorithm 1: RRT∗ algorithm. *AccCost*(c) returns the accumulated cost from the c_{start} to c which can be easily stored in the tree T. Note that the choise of the neighborhood is crucial for the optimality guarantee (see [7]) and that both the nearest neighbor and the neighborhood search can be implemented as efficient approximate searches without breaking the *asymptotic optimality* from [7].

2.2 Occupancy Mapping

One way to represent a 3D model is to discretize space into finite volumes that we call voxels and assign each voxel a set of properties describing it. Typically the space is discretized into a grid of equally sized cubes and each voxel bears a property called occupancy. The occupancy can be interpreted in two ways: 1) assuming that all voxels are either fully occupied (i.e. contain a solid object) or fully free the occupancy describes the probability $P(o)$ that a voxel is occupied, 2) it describes what fraction of the volume of a voxel is occupied (i.e. contains some solid object). For case 2) another property is usually assigned to each voxel that describes the number of observations or the certainty of the occupancy value. Here we will focus on interpretation 1) as it is commonly used in robotic applications. More details on interpretation 2) will be given in Sec. 5.4.1 where we will use it.

Representing an occupancy map as a dense grid of voxels requires memory that grows linear with the mapped volume which grows cubic with the length along the longest dimension of the mapped space. Mapping a space of 100m $\times$ 100m $\times$ 100m already requires 1 billion voxels (i.e. 1 GB if the occupancy is represented as a single-precision floating point number). In particular on mobile robotic platforms such memory requirements can already surpass the system capacity. However, when mapping typical scenes we observe that often large contiguous volumes are free, occupied or unobserved (i.e. share the same occupancy value). This allows the use of a hierarchical data structure instead of a dense grid of voxels. An octree is used as it implicitly captures the layout of the voxel grid and can represent large cubes with homogeneous occupancy values. Each node of the octree has an occupancy value and references to its eight children nodes. If the children nodes are uninitialized this means that all the volumes represented by the children share the same occupancy value that is stored in the node. Instead of saving a leaf (or size) property with each node we fix the maximum volume of the octree by specifying its voxel size and the maximum tree depth (i.e. 16). A node is then

simply identified as a leaf node if it is at the maximum tree depth. For a voxel size of $0.1m$ this allows for a volume of size approximately $6.5\text{km} \times 6.5\text{km} \times 6.5\text{km}$.

To map a scene with an occupancy map we start by initializing the voxels with an initial occupancy value of 0.5. We assume that we are given a depth image from a known camera pose with known intrinsics (otherwise we can compute the camera pose and depth image as described in Sec. 2.3). For each pixel of the depth image we iterate over all voxels that intersect with the ray starting from the projection center of the camera towards the 3D point p represented by the pixel. We update all voxels that the ray passes through in a probabilistic manner according to an inverse sensor model:

$$L(o|z) = \begin{cases} l_{\text{occ}} & \text{the 3D point } p \text{ is within the voxel} \\ l_{\text{free}} & \text{otherwise} \end{cases} \quad , \tag{2.8}$$

where $L(n|z)$ is the log odds ratio of a voxel being occupied given a single measurement z. We refer the reader to [9] for more details. By representing both the sensor model and the occupancy value as a log odds ratio (i.e. $L(o) = P(o)/\neg P(o)$), the update of the occupancy with a new measurement is a simple addition:

$$L(o|z_{1:t}) = L(o|z_{1:t-1}) + L(n|z) \quad , \tag{2.9}$$

where $L(o|z_{1:t})$ is the log odds ratio of the occupancy given t measurements $z_1, \ldots, z_t$. We follow [9] and use $l_{\text{occ}} = 0.85$ and $l_{\text{free}} = -0.4$ (which corresponds to a probability of 0.7 for a 3D point within the voxel and 0.4 otherwise).

2.3 3D Reconstruction

The process of generating a 3D model of an object or scene is called 3D reconstruction. Here we look at the 3D reconstruction from a set RGB images. This task is often split into two steps:

1. In the Structure from Motion step the camera pose of each image is computed together with a sparse set of 3D points in the scene.

2. In the Multi-View Stereo (MVS) process a disparity or depth map is generated for each image by computing and filtering pixel-wise correspondences with other images.

Finally, the resulting depth maps can be combined into a volumetric [9] or mesh-based [10] representation.

Notation Here we define some notation to ease the following discussion:

$p \in \mathbb{R}^3$ A point p in 3 dimensional Euclidean space.

$z \in \mathbb{R}^2$ An image point in 2 dimensional Euclidean space.

$v \in \mathbf{SE}(3)$ A camera viewpoint or pose v describes a rigid transformation in the 3 dimensional Euclidean space from world coordinate frame to the viewpoint coordinate frame. This is known as the extrinsic parameters of a camera view. It is commonly represented as a $\mathbb{R}^{4\times4}$ matrix or a unit quaternion together with a translation vector in $\mathbb{R}^3$. When represented as a 4×4 matrix M a point p coordinates can be transformed by matrix multiplication of M with its homogeneous representation.

$\mathcal{T}_v : \mathbb{R}^3 \to \mathbb{R}^3$ Transforms a point in world coordinate frame to the local coordinate frame of a camera with viewpoint v.

$\theta \in \mathbb{R}^K$ The intrinsic parameters θ of the camera. Depending on the camera model these can comprise focal lengths, principal points and distortion coefficients.

$\pi_\theta : \mathbb{R}^3 \to \mathbb{R}^2$ Projects a point in a camera local coordinate frame onto the image plane of the camera with intrinsic parameters θ.

Structure From Motion Structure from Motion is the process of extracting the camera poses v_j from a set of images. Here we give a high level description of the steps and refer to the literature for details [11, 12, 13, 14].

First we identify 2D keypoints z in each image and extract feature descriptors for each of them. These keypoints should be at corners of the gradient image so they can be found and localized in other images. Next we match feature descriptors between images. To enable this the ideal feature descriptor should be orientation- and scale-independent to enable matching of images from different viewpoints. Additionally, it is desirable that the descriptor is invariant to a homography so that we can match viewpoints that look from a different angle onto the surface of the keypoint. After matching features between two images we can compute a homography or a fundamental matrix relating the two images. This is done in a robust way, e.g. using RANSAC. By requiring a minimum number of inliers the resulting relation can be geometrically verified.

Now given all these relations the task is to recover the camera poses v_j. The resulting camera poses are only defined up to a similarity transform: The scale is ambiguous due to the projective geometry of a camera and there are 6 degrees of freedom for choosing a coordinate system. This is usually addressed by initially computing the relative pose of two images with a fixed baseline vector (3 rotation DOFs and 1 scale DOF) and putting one of the images at the origin of the coordinate system (3 translation DOFs). Choosing a good initial pair is a difficult problem in itself. The relative pose itself can be computed by decomposing the homography or fundamental matrix. Using the camera poses we can triangulate the common feature points to get a set of initial 3D points p_i. The initial poses v_j and 3D points p_i make up the initial reconstruction.

Based on this reconstruction further images can be added one by one by solving a Perspective-n-Point problem of the keypoints z in the new image and the corresponding 3D points p_i in the reconstruction. Again this is usually done in a robust way and geometrically verified. After computing the pose of the new image we can try to triangu-

late matching feature points without a corresponding 3D point. The reconstruction is then extended by the new pose and 3D points.

Bundle Adjustment When adding more and more images the estimated camera poses v_j will drift due to accumulation of errors of the triangulated 3D points p_i. To remedy this Bundle Adjustment is performed regularly with the growing reconstruction. In Bundle Adjustment all 3D points are projected into the images that observed them. These projections are the observations whereas the corresponding keypoints are the measurements. Bundle Adjustment itself is then the minimization of the residuals between observations and measurements where the the camera intrinsics, camera extrinsics and the 3D points are free variables to be optimized:

$$\mathcal{P}, \mathcal{V}, \theta = \text{argmin}_{\{\hat{p}_1,\ldots,\hat{p}_N\},\{\hat{v}_1,\ldots,\hat{v}_M\},\hat{\theta}} \sum_{i}^{N} \sum_{j}^{M} r^2_{i,j,\hat{p}_i,\hat{v}_j,\hat{\theta}} \tag{2.10}$$

$$r_{i,j,p,v,\theta} = \begin{cases} \pi_\theta(\mathcal{T}_{v_j}(p_i)) - z_{i,j} & \text{if 3D point } i \text{ is observed in view } j \\ 0 & \text{otherwise} \end{cases},$$

where N is the number of 3D points p_i, M is the number of camera viewpoints v_j, $z_{i,j} \in \mathbb{R}^2$ is the observed projection of 3D point p_i in camera view v_j and all camera views share the same intrinsic parameters θ. This optimization will distribute the accumulated errors between all camera poses and 3D points and result in a better and more robust reconstruction. To further increase robustness against outliers the squared residual is often replaced with a robust loss function.

Multi-View Stereo Given camera poses of multiple images we want to compute a disparity or depth map for each of these images. This is known as Multi-View Stereo and is a generalization of dense stereo matching to multiple images. Here we provide a quick overview and refer the reader to the literature for details [11, 12, 15, 16, 17, 18].

In dense stereo matching we are given two images A, B observing the same scene and their relative pose. The goal is to compute for one or both images a disparity value or depth value for each pixel. Often the first step in the process is to rectify both images, i.e. project both images onto a common image plane, such that epipolar lines are horizontal and at the same vertical position in the other image. For each pixel in image A we can search for a matching pixel in image B within a window of no and maximum disparity. In block matching stereo the determination of pixel matches is done by computing a similarity measure between small windows around the pixels. To compute this similarity a key assumption in many methods is that of fronto-parallel patches, i.e. the patch is assumed to be parallel to the image plane. Many similarity measures can be used such as sum of absolute differences, sum of squared differences or normalized cross-correlation but also rank-based measures or learned measures [19, 20].

We note here that the stereo matching problem is an ill-posed problem. This can be easily seen for the case of two images of a white homogeneous surface where matching is ambiguous. Another fundamental problem is that of handling pixels in image A that are occluded in image B. To reduce these ambiguities one can post-process the resulting disparity maps. Another option is to introduce prior information, such as smoothness assumptions, is to use global methods such as graph cuts or dynamic programming that reason over more than one pixel. These methods typically work on a cost volume were the similarity is computed for each pixel and each disparity.

The problems of occlusions and ambiguities in the stereo matching of two images can be remedied by incorporating more images. These additional images might feature different occlusions or disambiguate multiple hypothesis of a pixel disparity. This is often done by filtering along the multiple depth maps that were computed. Another approach is to compute matching cost volumes for multiple images in inverse depth space. Filtering can then be performed on these cost volumes and can provide additional robustness. Instead of the filtering approach one can also select views that are most promising for depth map computation [21, 22]. Another successful paradigm devi-

ated from the depth map based reconstruction and instead tried to reconstruct a set of 3D patches that is consistent over multiple views [15]. In recent state-of-the art methods the patch-match framework [23] was adapted to the stereo matching problem with great success [22, 24, 17, 18]. In this framework we start with a randomly initialized or uninitialized depth map and insert depth values from sparse 3D points that were computed in the Structure from Motion step. In each iteration depth values from pixels are propagated to their neighbors and overwrite their depth values if this reduces the matching cost. After the propagation new depth values are sampled at each pixel and overwrite the previous depth value if the new depth value reduces the matching cost. Due to the parallel nature of the framework it allows for efficient implementations on modern GPUs. In addition to the depth estimates one can also incorporate normal estimates for each pixel to handle non-fronto-parallel surfaces. Going even further the patch-match framework can be extended to perform pixel-wise view selection by formulating a probabilistic model that is approximately inferred with each patch-match iteration [24, 14].

2.4 Submodular Optimization

Submodular optimization is common in the field of sensor placement where the task is to find the best sensor configuration according to some metric with respect to a certain budget, e.g. a maximum number of sensors [1]. The property *submodularity* refers to a diminishing returns property that occurs in many tasks. This means that the selection of a sensor will provide less and less additional use on top of the already selected sensors as their number grows. To give a formal definition let S be a set of possible sensor configurations, i.e. the positions where a sensor can be deployed. A function $f : 2^S \to \mathbb{R}$ is called *submodular* iff it fulfills the following property:

$$\begin{aligned} &\text{For every} A, B \in 2^S \text{ with } A \subseteq B \text{ and } x \in S : \\ &f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) \quad . \end{aligned} \tag{2.11}$$

Many tasks in sensor placement and related areas can be formulated as an optimization problem over sets:

$$X = \arg\max_{X \subseteq S}(f(X)) \quad \text{such that } b(X) \geq 0 \quad , \tag{2.12}$$

where the objective function f specifies how good a selection of sensors is and the function $b : 2^S \to \mathbb{R}$ is a constraint on the selection of sensors. A common example for b is a cardinality constraint $b(X) = N - |X|$ requiring that at most $N \in \mathbb{N}$ sensors are selected.

Greedy Selection In general the optimization problem in 2.12 is NP-complete [1], however, if the objective function in 2.12 is *submodular* many approximate solution methods have been proposed that often provide a theoretical guarantee on the optimality of the approximate solution. One prominent example for the problem with a cardinality constraint is the greedy selection of sensors based on their incremental improvement Δf of f:

$$\Delta f(x, Y) = f(X_i \cup \{x\}) - f(X_i) \quad . \tag{2.13}$$

The greedy selection is then recursively defined as:

$$\begin{aligned} X_0 &= \emptyset \\ x_{i+1} &= \arg\max_{x \in S \setminus X_i} \Delta f(x, X_i) \\ X_{i+1} &= X_i \cup \{x_{i+1}\} \quad . \end{aligned} \tag{2.14}$$

The approximate solution is then given by $X_N \in 2^S$ and we are guaranteed that

$$f(X_N) \geq (1 - 1/e) f(X^*) \quad , \tag{2.15}$$

where $X^* \in S^2$ is the optimal solution [25, 1].

Lazy Evaluation It is easy to see that the number of evaluations of f required for the greedy algorithm is $O(N|S|)$ as we need to greedily select N elements and for each selection we need to evaluate the function f for $\mathrm{O}(|S|)$ different elements. Unfortunately, in many applications the set $|S|$ is large and the evaluation of f is expensive. However, we can speed up the computation in many cases by exploiting the *submodular* property of f. To this end we keep a list L_i of the elements in S such that the elements are sorted in descending order according to $\Delta f(\cdot, X_i)$. Given the sorted list L_i it is easy to select the next greedy element as

$$x_{i+1} = L_0 \tag{2.16}$$

as the element with the highest incremental improvement Δf is at the beginning of the list. The important observation is now that we only need the first element in the list to be the one with the highest Δf value and we can compute the next list L_{i+1} with this property without recomputing all Δf values. Recall that the *submodular* property tells us that $\Delta f(x, X_{i+1}) \leq \Delta f(x, X_i)$ as $X_i \subset X_{i+1}$. We can now look at the first element in the list, compute the new value $\Delta f(x, X_{i+1})$ and move the element along the list until the next element has a smaller value. We do not need to look at the following values as they can only have decreased. We repeat this until the first element does not have to be moved and obtain a list L_{i+1} where the first element is the one with the highest Δf. In many practical applications this lazy evaluation can save computational costs of an order of magnitude or more.

Orienteering The problem when one is not only interested in a selected set of sensors $X \subseteq S$ but also in a route that connects those sensors is called *orienteering* [26]. The connectivity between sensors is modeled by a graph $G = (S, E)$ where E are the edges connecting sensors. We can still express the problem with (2.12) by incorporating the requirement of having a route along the sensors in X into the constraint $b(X) \geq 0$. In most instances of *orienteering* there is a max-

imum budget $L_{max} \in \mathbb{R}$ that can be used. The constraint function $b(X)$ can then be written as

$$b(X) = L_{max} - L(X) \tag{2.17}$$

$$L(X) = \begin{cases} \text{Length of shortest route of } X & \text{if route exists} \\ \infty & \text{if no route exists} \end{cases}. \tag{2.18}$$

As in the set cover problem described above there exist many approximate solution approaches for the *orienteering* problem where the cost function is *submodular* [27, 28, 29, 30]. Typically these methods are still impractical for the problems considered in this thesis or require certain properties of the cost function, such as strong locality, that do not hold for vision based systems which provide long sensing ranges.

2.5 Artificial Neural Networks

Artificial neural networks (ANNs) have become the default choice of machine learning models for image related and recently also 3D data related tasks. Here we give a very coarse idea of these models as they are relevant for the content presented in Chapter 5. For a comprehensive introduction and overview we refer the reader to the excellent *Deep Learning* book by Goodfellow et al. [31].

The key idea of ANNs is to use a series of differentiable operations to transform the input data to the desired output in a non-linear way. Some of these operations are parametrized. By also defining a differentiable loss on this output we can use automatic differentiation to compute the gradient of the loss with respect to the parameters and perform stochastic gradient descent to decrease the loss. Although the relationship of loss and parameters is typically highly non-convex empirically good local minima can be found by ensuring suitable initialization of the parameters and using well-tested combinations of

operations. One notable and simple operation is the matrix multiplication of inputs with a weight matrix. This is also known as a fully connected layer and often includes the addition of a bias vector. It is usually followed by a non-linearity such as a *tanh*, *sigmoid* or rectified linear unit ($ReLU$). However, when multiple fully connected layers are used on image data and the intermediate representations are also large the number of parameters in the weight matrix quickly explode. An alternative and very important operation for image or volumetric 3D data is the convolution (or correlation) with a weight kernel which has only a small number of parameters (depending on the kernel size) and already provides an often desired translational invariance.

Chapter 3

Optimization-Based Planning of Quadrotor Trajectories

This chapter is based upon our work in [32]. Here we propose a computational design tool that allows end-users to create advanced quadrotor trajectories with a variety of application scenarios in mind. Our algorithm allows novice users to create quadrotor based use-cases without requiring deep knowledge in either quadrotor control or the underlying constraints of the target domain. To achieve this goal we propose an optimization-based method that generates feasible trajectories which can be flown in the real world. Furthermore, the method incorporates high-level human objectives into the planning of flight trajectories. An easy to use 3D design tool allows for quick specification and editing of trajectories as well as for intuitive exploration of the resulting solution space. We demonstrate the utility of our approach in several real-world application scenarios, including aerial-videography, robotic light-painting and drone racing.

3.1 Introduction

In recent years micro-aerial vehicles (MAVs), in particular Quadrotors, have seen a rapid increase in popularity both in research and the consumer mainstream. While the underlying mechatronics and control aspects are complex, the recent emergence of simple to use hardware and easy programmable software platforms has opened the door to widespread adoption and enthusiasts have embraced MAVs such as the AR.Drone or DJI Phantom in many compelling scenarios including aerial photo- and videography. Furthermore, the HCI community has begun to explore these drones in interactive systems such as sports assistants [33, 34, 35] or display [36] of content.

Clearly there is a desire to use such platforms in a variety of application scenarios. Current SDKs already give novices access to manual or waypoint based control of MAVs, shielding them from the underlying complexities. However, this simplicity comes at the cost of flexibility. For instance, flying a smooth, spline-like trajectory or aggressive flight maneuvers, for example to create an aerial light show (e.g., [37, 38]), is tedious or impossible with waypoint based navigation. These limits exist because manufacturers place hard thresholds on the dynamics to ensure flight stability for inexperienced pilots.

More importantly, state-of-the-art technologies offer only very limited support for users who want to employ MAVs to reach a certain high-level goal. This is maybe best illustrated by the most successful application area – that of aerial videography. What a few years ago was limited to professional camera crews, requiring cost-intensive equipment like a helicopter, can now, in principle, be done by end-users with a MAV and an action camera. However, producing high-quality aerial footage is not an easy task – it demands attention to the creative aspects of videography such as frame composition and camera motion (cf. [39]). In the case of airborne cameras, an operator needs to fly smoothly, accurately and safely around a camera target. Furthermore, the target has to be framed properly alongside further creative considerations. Thus this is a difficult task and typically requires at least two experienced operators – one pilot and a camera

man (cf. [40]). Our method tackles this problem by enabling a single novice user to fly challenging trajectories and still create aesthetically pleasing aerial footage.

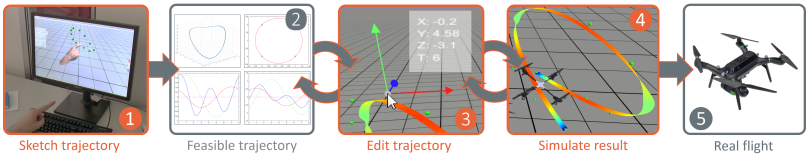

Figure 3.1: System workflow schematically. (1) User sketches keyframes. (2) An optimization method generates a feasible trajectory. (3+4) The user can quickly iterate over the trajectory and explore the solution space of feasible trajectories via a physics simulation or a rendered preview (see Fig. 3.2, D). (5) Final trajectory can be flown with a real MAV.

3.1.1 Overview & Contribution

Embracing the above challenges we propose a computational method that enables novice end-users to create quadrotor use-cases without requiring expertise in *either* low-level quadrotor control *or* specific knowledge in the target domain. The core contribution of our method is an optimization-based solution that generates *feasible trajectories* for flying robots while taking *high-level user goals* such as visually pleasing video shots, optimal racing trajectories or aesthetically pleasing MAV motion into consideration. Furthermore, we develop an easy-to-use tool that allows for straightforward specification of flight trajectories and high-level constraints. Our approach guides the users in exploring the resulting design space via a 3D user interface and allows for quick iteration until finding a solution which fits best with the user's intentions.

We demonstrate the flexibility of our approach in three real-world

scenarios including aesthetically pleasing aerial-videos, robotic light-painting and drone racing.

3.2 Related Work

3.2.1 MAVs in HCI

With MAVs becoming consumerized the HCI community has begun to explore this design space. FollowMe [41] is a MAV that follows a user and detects simple gestures via a depth camera, whereas others have proposed using head motion for MAV control [42], while [36] propose a simple, remote controlled flying projection platform. Several setups have been proposed that turn such MAVs into flying, personal companions. For example, to act as jogging partner [34] or general purpose sports coach [33], or as an actuated and programmable piece of sports equipment [35].

Commercially available drones, targeted at the consumer market, shield the user form low-level flight aspects and provide simple manual control (e.g. using smartphones as controller) or waypoint based programmatic navigation as well as GPS based person following. This dramatically lowers the entry barrier for novices but also limits the ceiling of achievable robotic behavior. Our approach also aims for simplicity but gives more power to the users, enabling even novices to design and implement complex flight trajectories, concentrating on the high-level goals of the application domain.

3.2.2 Video Stabilization & Camera Path Planning

Improving the visual quality of end-user produced content is a goal we share with post production video stabilization. Inspired by early work which formulates the problem and discusses the aesthetics of cinematography [43] a number of approaches employ computer vision methods to estimate the original, jerky camera path. Based on this a new, smooth path is computed to generate stabilized video [44, 45] and

even time-lapse footage [46] from the source material. Camera path planning has also been studied extensively in the context of virtual environments using constraint based [47, 48] or probabilistic [49] methods. However, these methods are not limited by real-world physics and hence can produce arbitrary camera trajectories and viewpoints. Our approach differs from the above as we propose a *forward* method that gives the user full control over the creative aspect of camera planning while simultaneously optimizing for physical feasibility of the flight path and cinematographic objectives. A 3D simulation lets the user explore the design space before flying the actual trajectory and hence helps in understanding the trade-offs to consider.

3.2.3 Computational Design

Sharing the goal of unlocking areas that previously required significant domain knowledge to novice users, the HCI and graphics communities have proposed several methods that give novice users control over aesthetic considerations while achieving functionality. Recent examples include digitally designed gliders [50] and kites [51] with optimized aerodynamic properties. At the core of these approaches are sophisticated simulations or analytical models of the problem domain that carefully balance accuracy and rapid responses to ensure interactivity while maintaining guarantees (e.g., physical stability). We build on domain knowledge from the robotics and MAV literature and propose an interactive design tool for complex MAV behavior usable by non-experts.

3.2.4 Robotic Behavior and Trajectory Generation

Automating the design of robotic systems based on high-level functional specifications is a long-standing goal in robotics, graphics and HCI. Focussing on robot behavior only, simple direct touch and tangible UIs [52], and sketch based interfaces to program robotic systems [53, 54] have been proposed. Visual markers have been used to control robots explicitly, for example as kitchen aides [55], or implicitly

[56, 57], to schedule tasks for robots in-situ which are then collected and executed asynchronously.

Generating flight trajectories for MAVs is well-studied in robotics. In particular, the control aspects of aggressive and acrobatic flight is an active area of research (e.g., [58]). Mellinger et al.'s work on generating minimum snap trajectories [59] is the most related to ours. While they specify a trajectory as a piecewise polynomial spline between keyframes, we discretize the trajectory into small piecewise linear steps. The result is a more intuitive formulation of the optimization problem, making the incorporation of additional constraints and objectives much easier. We extend the approach in [59] by optimizing trajectories for flyability *and* for high-level human objectives. We place the users in the loop and provide easy-to-use tools to design quadrotor trajectories according to high-level objectives.

Joubert et al. [60] share a similar goal in proposing a design tool that allows novice users to specify a camera trajectory, simulate the result, and execute the motion plan. In contrast to our work, their method does not automate feasibility checking but delegates the correction of violations to the user. Furthermore, our method allows to treat keyframes as soft instead of hard constraints, allowing to trade off feasibility against keyframe matching. We also incorporate a larger number of high-level user constraints, such as additional cinematographic goals and collision-free trajectories, into the algorithm – requiring a different formulation of the optimization problem. Finally, we demonstrate the gain in generality in our approach in the additional use cases of light writing and aerial racing.

3.3 Notation

$m \in \mathbb{R}$	Quadrotor mass.
$I_\psi \in \mathbb{R}$	Quadrotor moment of inertia about body z-axis.
$\mathbf{r} \in \mathbb{R}^3$	Quadrotor center of mass.
$\psi \in \mathbb{R}$	Quadrotor yaw angle along world z-axis.
$\mathbf{F} \in \mathbb{R}^3$	Force acting on the quadrotor center of mass.
$\mathbf{g} \in \mathbb{R}^3$	Gravity force aligned with the world z-axis.
$M_\psi \in \mathbb{R}$	Torque acting on the quadrotor along the body z-axis.

3.4 System Overview

We propose an end-to-end system that allows users to generate motion plans for quadrotors that are 'flyable' and adhere to high-level human-specified objectives for a variety of application scenarios. Fig. 3.1 illustrates the design process.

Using for example a LeapMotion controller the user specifies keyframes, each consisting of a position and a time (1). The optimization algorithm generates an initial 'flyable' trajectory from these inputs, i.e., one that lies within the physical capabilities of the underlying quadrotor hardware (2). The method aims to find a solution that goes through all specified keyframes, however the user may now adjust both a keyframe's position and timing as well as other parameters such as the overall flight time, the optimization's objective (e.g., minimization of velocity) or the extend to which the generated trajectory should follow the optimization's objective versus the position of the specified keyframes (3). This can result in trajectories that do not directly meet the user inputs but are the best trade-off between the potentially conflicting use-case specific constraints. A built-in physics simulation allows the user to virtually fly the quadrotor and thus provides a better understanding of the expected real-world behavior and enables rapid iteration of trajectories (4). This tool already enables

the design of various flight-maneuvers for example designing an aerial race-course or a light-show (please see video for an illustration of the design process and results).

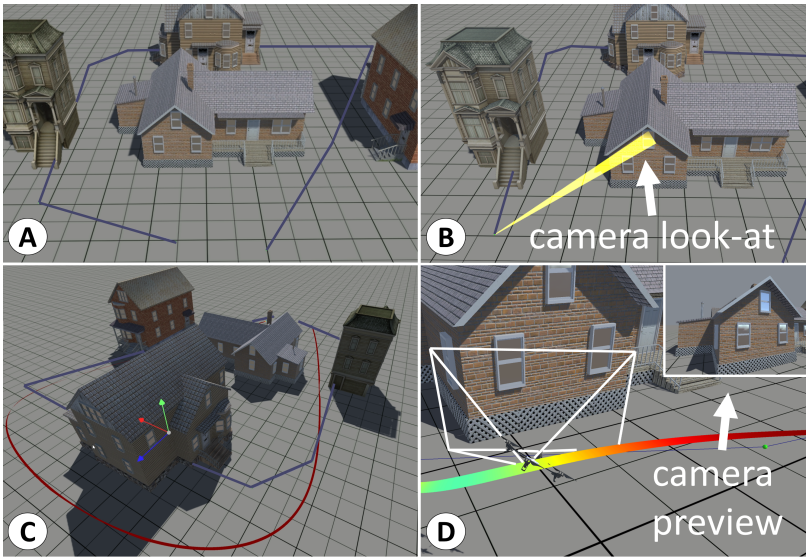

Figure 3.2: Planning of aerial video shots. (A) User specifies sparse keyframe positions connected by straight lines (purple). (B) For each keyframe the user also specifies a desired camera target (yellow). (C) We generate a smooth and collision free motion plan alongside gimbal control inputs (dark red). (D) Virtual preview allows for rapid prototyping showing the current camera frustum and a camera preview.

However, the goal of our work is to enable more complex end-user scenarios as well. To this end we have extended our method to also integrate high-level aesthetic constraints that are not necessarily di-

rectly associated with the basics of quadrotor control. Fig. 3.2 illustrates how our tool can be used to plan aerial videography shots. In this case, the user designs an initial camera trajectory around one or several targets. In addition to the keyframes the user specifies targets which shall be captured by the on-board camera (Fig. 3.2, B). Our algorithm generates both a quadrotor trajectory and a gimbal trajectory within the physical bounds. To acquire visually pleasing footage our method incorporates cinematographic constraints such as smooth camera and target motion, smooth transitions between multiple targets and reduction of perspective distortions. Furthermore, the algorithm takes obstacle information into account and automatically routes the trajectory through free-space (see Fig. 3.2, C). It would also be straightforward to integrate other constraints such as limits of the coverage of a tracking system or government flight regulations.

In order to better understand the implications of the camera planning our tool allows the user to virtually fly the shot by dragging the virtual quadrotor along the trajectory (Fig. 3.2, D). For each point in time the tool renders the scene as it would be captured in reality. The user can then edit the plan and iterate over different alternatives. Once satisfied the generated trajectories for quadrotor and gimbal can be deployed as a reference to be followed by a real quadrotor.

3.5 Method

So far we have discussed the proposed design tool at a high-level and focused on how the user accomplishes certain tasks.

We now introduce the underlying method we use to generate trajectories. To be able to reason about flight plans computationally, a model of the quadrotor and its dynamics are needed. This is a complex and challenging topic and we refer the reader to the Appendix for the full non-linear model that is needed to control the position and dynamics of the robot during flight (please see also [61]). The full model directly relates the inputs of a quadrotor to its dynamics – this however makes trajectory generation a challenging problem and inte-

grating such a highly non-linear model into an optimization scheme is complicated, incurs high computational cost and negates convergence guarantees [59]. However, for most application scenarios considered here a full non-linear treatment is not necessary as demonstrated by our results. In particular if the goal is to generate trajectories only (i.e., position and velocities) rather than the full control inputs as in [59]. Therefore, we present a linear approximative model of the quadrotor and detail the optimization-based algorithm based on it.

3.5.1 Approximate Quadrotor Model for Trajectory Generation

When generating a trajectory we want to ensure that it can be followed by a quadrotor, i.e. a flight plan where each specified position and velocity can be reached within the time limits without exceeding the limits of the qudrotors inputs.

Therefore, we chose to approximate the quadrotor as a rigid body, described by its moment of inertia only along the world frame z-axis (i.e. we ignore pitch and roll of the quadrotor):

$$\begin{aligned} m\ddot{\mathbf{r}} &= \mathbf{F} + m\mathbf{g} \in \mathbb{R}^3 \\ I_\psi \ddot{\psi} &= M_\psi \in \mathbb{R}, \end{aligned} \tag{3.1}$$

where $\mathbf{r}$ describes the center of mass of the quadrotor, ψ is the yaw angle of the quadrotor, m is the mass of the quadrotor, I_ψ is the moment of inertia about the body-frame z-axis, $\mathbf{F}$ is the the force acting on the quadrotor and M_ψ is the torque along the z-axis. This approximation allows to generate trajectories in the *flat output* space of the full quadrotor model (see Appendix for more details).

In addition to the equations of motion we introduce bounds on the maximum achievable force and torque:

$$\mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max} \in \mathbb{R}^4, \tag{3.2}$$

where $\mathbf{u} = [\mathbf{F}, M_\psi]^T$ is the input of the system.

With this model it is not possible to exploit the full dynamic agility of a quadrotor. As an example, consider the situation of accelerating straight upwards by rotating all motors at maximum speed. To now also rotate around the body-frame z-axis we would have to lower the speed of motors 2 and 4, reducing the total thrust of the quadrotor. Currently we do not incorporate this coupling between the translational and rotational dynamics into the bounds Eq. (3.2) of our approximate linear model Eq. (3.1). Therefore, to still ensure that a quadrotor can follow trajectories generated on base of this approximation conservative bounds are required. Nonetheless, our results and applications demonstrate that these bounds still allow the quadrotor's agility to be sufficiently rich for many use cases. We refer the interested reader to the Appendix for details on how to choose these bounds.

For trajectory generation we rewrite the approximate model as a first-order dynamical system and discretize it in time with a time-step Δt assuming a zero-order hold strategy, i.e. keeping the inputs constant in between stages:

$$\mathbf{x}_{i+1} = A_d \mathbf{x}_i + B_d \mathbf{u}_i + c_d, \tag{3.3}$$

where $\mathbf{x}_i = [\mathbf{r}, \psi, \dot{\mathbf{r}}, \dot{\psi}]^T \in \mathbb{R}^4$ is the state and $\mathbf{u}_i$ is the input of the system at time $i\Delta t$. The matrix $A_d \in \mathbb{R}^{8x8}$ propagates the state $\mathbf{x}$ forward by one time-step, the matrix $B_d \in \mathbb{R}^{8x4}$ describes the effect of the input $\mathbf{u}$ on the state and the vector $c_d \in \mathbb{R}^8$ that of gravity after one time-step.

3.5.2 Trajectory Generation

With this approximate quadrotor model in place we can now discuss the optimization scheme to generate trajectories. The user specifies M keyframes describing a desired position k_j at a specific time-point $\eta(j)\Delta t$, where $\eta : \mathbb{N} \to \mathbb{N}$ maps the index of the keyframe to the corresponding time-point. In the case of mouse-based user input we assume constant time between consecutive positions. To compute a

feasible trajectory over the whole time horizon $[0, t_f]$ we discretize time with a time-step Δt into N stages. The variables we optimize for are the quadrotor state x_i and the inputs u_i of the system Eq. (3.3) at each stage $i\Delta t$.

The first goal of our optimization scheme is then to follow the user inputs as closely as possible, expressed by the cost

$$E^k = \sum_{j=1}^{M} ||r_{\eta(j)} - k_j||^2. \tag{3.4}$$

A small residual of E^k indicates a good match of the planned quadrotor position and the specified keyframe. The bounds Eq. (3.2) together with Eq. (3.3) and Eq. (3.4) can then be formulated as a quadratic program

$$\begin{aligned} \underset{X}{\text{minimize}}\ & \frac{1}{2} X^T H X + f^T X \\ \text{subject to}\ & A_{ineq} X \leq b_{ineq} \\ \text{and}\ & A_{eq} X = b_{eq}, \end{aligned} \tag{3.5}$$

where X denotes the stacked state vectors x_i and inputs u_i for each time-point, H and f contain the quadratic and linear cost coefficients respectively which are defined by Eq. (3.4) , A_{ineq}, b_{ineq} comprise the linear inequality constraints of the inputs Eq. (3.2) and A_{eq}, b_{eq} are the linear equality constraints from our model Eq. (3.3) for each time-point $i \in 1, \ldots, N$. Note that this optimization problem is a specialization of the general trajectory generation problem Eq. (2.2) described in Section 2.1. Problem Eq. (3.5) has a sparse structure and can be readily solved by most optimization software packages. However, this problem is ill-posed and the result for a particular set of keyframes might be counterintuitive at first. Since we only measure the match of quadrotor position at the keyframe times the state at other time-points is not constrained in any way except for the quadrotor dynamics. Therefore a straight path between two keyframe positions is as good as a zig-zag pattern if it is feasible. An example

of this is shown in Fig. 3.4. To attain better results we have to further regularize the solution.

In many robotics application one goal is to minimize energy expenditure and this is often done by penalizing non-zero inputs or in other words attempting to reach desired positions with minimal wasted effort. For end-user applications, for example in the context of a racing game, one can also aim to attain smooth trajectories by penalizing higher derivatives of the quadrotor's position with respect to time such as acceleration (2nd) or jerk (3rd). We introduce the cost

$$E^d = \sum_{i=q}^{N} ||D^q \begin{bmatrix} x_i \\ \dots \\ x_{i-q} \end{bmatrix} ||^2, \tag{3.6}$$

where D^q is a finite-difference approximation of the q-th derivative from the last q states. Since the term jerk is not commonly known outside of engineering fields an intuition is to think of high values of jerk as a feeling of discomfort caused by too sudden motion. Humans tend to plan motion by minimizing the norm of jerk [62] and thus, minimizing jerk results in motion plans that appear pleasant to a human.

The combined cost $E = \lambda_k E^k + \lambda_d E^d$ with weights $\lambda_{k|d}$ is still a quadratic program and enables us to generate trajectories that are feasible and that are optimal in the sense of Eq. (3.5). While still relatively basic in functionality this already enables a variety of use-cases such as aerial light-shows and racing-games as illustrated in the next section.

3.5.3 Optimizing for Human Objectives

With the basics in place we now turn our attention to including high-level human objectives into the optimization. As a running example we will consider the task of planning an aerial video-shot but we would like to emphasize that many other tasks such as 3D reconstruction or

projector based augmented reality could be implemented in the same way.

We have already discussed how this process works from the user's perspective in the system overview. Here the user provides additional camera targets that should be recorded at a specific time (see Fig. 3.2). Furthermore, we assume that the quadrotor is equipped with a gimbal that we can control programmatically. From a cinematographic standpoint, the most pleasant viewing experience is conveyed by the use of either static cameras, panning ones mounted on tripods or cameras placed onto a dolly (cf. [39]). Changes between these shot types can be obtained by the introduction of a cut or jerk-free transitions, i.e. avoiding sudden changes in acceleration. Furthermore, it is desirable to introduce saliency constraints or in other words we want not only the camera path to be smooth but also want to keep the target motion within the image frame as steady as possible and constrain it's motion to smooth motion.

To achieve these high-level objectives, we include a target position for each stage into the optimization variable. Analogous to the quadrotor position we introduce a cost term E^t that measures the deviations of user-specified keytarget points from the target positions at the corresponding stages. We penalize higher temporal derivatives (acceleration and jerk) of the target position by including finite differences in the cost term $E^{t,d}$. To link the quadrotor and target trajectories we introduce a simple gimbal model:

$$\begin{aligned}
\dot{\psi_g} &= u_{g,\psi} \\
\dot{\phi_g} &= u_{g,\phi} \\
[\psi_{g,min}, \phi_{g,min}]^T \leq [\psi_g, \phi_g]^T &\leq [\psi_{g,max}, \phi_{g,max}]^T \\
\mathbf{u}_{g,min} \leq [u_{g,\psi}, u_{g,\phi}]^T &\leq \mathbf{u}_{g,max},
\end{aligned}$$

where the inputs $u_{g,\psi}$, $u_{g,\phi}$ represent the angular velocities of the yaw ψ_g and pitch ϕ_g of the gimbal and both the inputs and the absolute angles are bounded according to the physical gimbal. The bounds specify the limits on the absolute angles and the angular velocities.

To ensure a smooth motion of the gimbal we introduce a cost E^g on temporal finite differences of the yaw and pitch angles analogous to Eq. (3.6). We do not incorporate the attitude of the quadrotor into our gimbal model and therefore the bounds have to be chosen conservatively.

The angle between the current camera direction $\mathbf{p_l}$ and the direction of the target $\mathbf{p_d}$ is depicted in Fig. 3.5. The error is then computed by

$$\alpha_{err} = \cos^{-1}\left(\frac{\mathbf{p_d} \cdot \mathbf{p_l}}{|\mathbf{p_d}||\mathbf{p_l}|}\right) \tag{3.7}$$

$$\mathbf{p_d} = \mathbf{r_t} - \mathbf{r} \text{ and } \mathbf{p_l} = \begin{bmatrix} \cos\phi_g \cos(\psi + \psi_g) \\ \cos\phi_g \sin(\psi + \psi_g) \\ \sin\phi_g \end{bmatrix},$$

where $\mathbf{r}$ is the quadrotor position, $\mathbf{r_t}$ is the target position and ψ is the pitch angle of the quadrotor.

Deviations of the camera direction from the desired target are penalized by

$$E^c = \sum_{i=1}^{N} \left(\alpha_{err}^i\right)^2, \tag{3.8}$$

where α_{err^i} is the camera angle error at stage i. Here the separation of target trajectory from the camera direction might seem surprising but it gives more flexibility as the user can choose the weights of the importance of target keypoints and the camera direction separately.

The final aesthetic cost is related to perspective effects. Viewpoints that are to high or low relative to the recorded object of interest lead to skew and results in strong vanishing lines in the image. This is illustrated in Fig. 3.6. While this effect maybe desired in some situations (imagine an overhead shot) we allow the user to supress

these types of distortions by optionally including a skewness cost E^s:

$$s_{err} = \frac{b_1}{b_2} - 1 = \frac{(\mathbf{p_d} + \tilde{\mathbf{h}}/\mathbf{2}) \cdot \mathbf{p_d}}{(\mathbf{p_d} - \tilde{\mathbf{h}}/\mathbf{2}) \cdot \mathbf{p_d}} - 1$$

$$\tilde{\mathbf{h}} = \begin{cases} \mathbf{h}, & \text{if } p_{d,3} >= p_{t,3} \\ -\mathbf{h}, & \text{else} \end{cases},$$

$$E^s = \sum_{i=1}^{N} \left(s_{err}^i\right)^2 \tag{3.9}$$

(3.10)

where $\mathbf{h}$ is a vertical vector pointing from the center of the target to the upper edge of the bounding box and s_{err}^i is the skewness error at stage i. In the computation we distinguish the case of a quadrotor flying above the target and the case of flying below a target.

Summing up the individual cost terms gives results in the final cost $E = \sum_{i=\{k,s,t,g,c\}} \lambda_i E^i$ Unfortunately α_{err} and s_{err} are non-linear in the variables of the motion plan and in consequence minimizing E can no longer be written as a quadratic program. We describe how we minimize E in the implementation section.

By penalizing snap of the quadrotor position and jerk of the camera motion the combined cost results in aesthetically pleasing footage (see the accompanying video). We can now generate a motion plan for a quadrotor that follows a target trajectory with the camera. To further support novice users we included an approximate collision-free scheme that can be used to keep a minimum distance from the target or stay at a safe distance from obstacles. Again we refer to the implementation section for details. Note that this only works for static objects where the position is known at the time of trajectory generation.

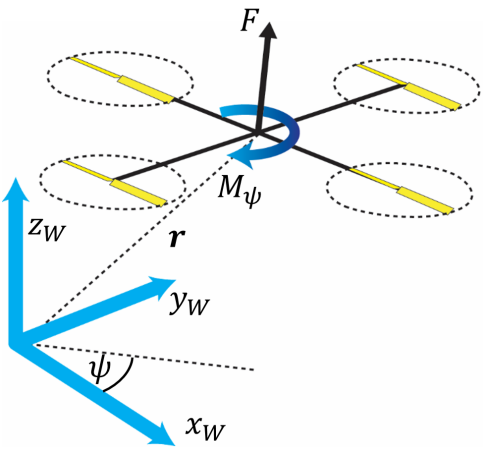

Figure 3.3: Our approximated quadrotor model with position $\mathbf{r}$, yaw angle ψ), world frame ($\mathbf{x}_W$, $\mathbf{y}_W$, $\mathbf{z}_W$), the moment acting on the quadrotor along the world frame z-axis M_{yaw} and the force $\mathbf{F}$ acting on the center of mass of the quadrotor.

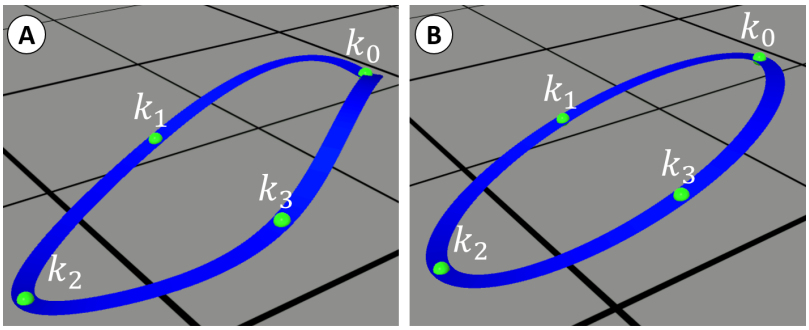

Figure 3.4: Same trajectory, optimized to only follow the keyframes (A) and to follow keyframes as well as minimizing snap on each stage of the optimization problem (B).

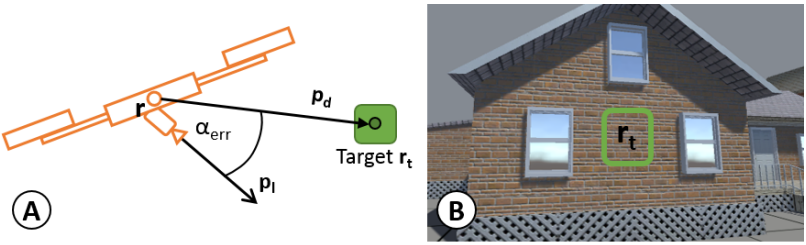

Figure 3.5: (A) camera direction $\mathbf{p_l}$ and distance $\mathbf{p_d}$. (B) effect of minimizing camera angle error α_{err} w.r.t. the target $\mathbf{r_t}$ in the center of the FOV of the camera.

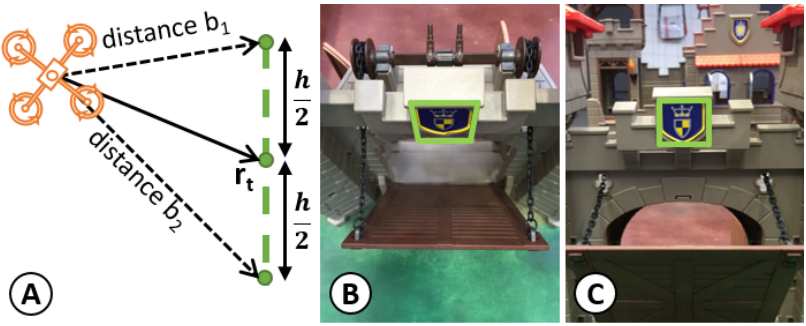

Figure 3.6: (A) Illustration of skewness error, where b_1 and b_2 are the distances to the upper and lower edge of the target bounding box. (B+C) Perspective without skewness correction (B) and with (C). Note that target is centered in both images.

3.6 Implementation

In this section we describe how we implemented the different components of our system. We start with describing the iterative quadratic programming scheme, then explain the onboard controller and the quadrotor hardware and finally show how we realized the design tool.

Iterative Quadratic Programming To solve the non-linear problem described above we resort to a scheme of iterative quadratic programming (IQP). The general idea is to linearly or quadratically approximate the problem around the current estimate of the solution. This approximate system is then solved and a better, consistent estimate of the solution is found. These iterative schemes usually converge within a few iterations despite the cost functions not being convex anymore. In our concrete implementation we start with an initial guess of the trajectory by interpolating the quadrotor positions and the camera targets between the keyframes. We also enforce all initial equality constraints to be fulfilled. As the proposed energies are usually non-convex a good initial guess is important to find a good solution. For each major iteration of our solver we build the H and f matrices of a quadratic program. This is done by quadratically approximating each of the cost terms around the trajectory X, note that this does not affect the quadratic terms in E. We also assemble the bounds and equality and inequality constraints and linearize them analogously. The fully assembled system is a sparse quadratic program and can be solved by most optimization packages. The solution gives us a change dX of the current motion plan. We perform a line search with the step length α to find a new motion plan $X_{next} = X + \alpha dX$. We lower α until we find a X_{next} with a cost $C(X_{next}) < C(X)$. This step is necessary as the cost of the approximated quadratic program is only an approximation of the real residual. An empirically derived serves as termination criteria.

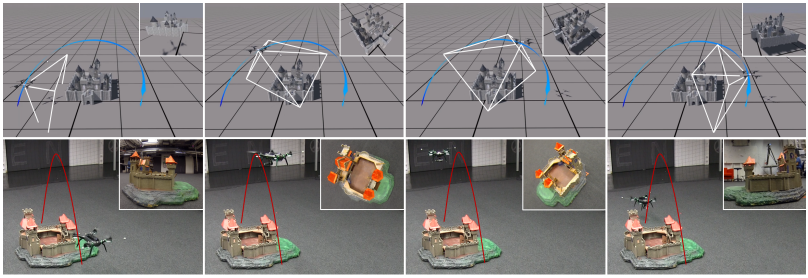

Figure 3.7: Aerial camera shot of a toy castle. Top row: planned trajectory in our design tool. Bottom row: flown trajectory.

Obstacle Avoidance: We approximate each obstacle as a static sphere with a radius o_r and introduce a non-convex constraint $||r \geq o_r||^2$. We linearize these constraints in each IQP iteration for each stage in time around the current trajectory X. Although we cannot guarantee global optimality of the resulting trajectories, this approach can be helpful for planning trajectories in scenes with known geometry and many objects. More advanced collision avoidance schemes, potentially taking dynamic targets into consideration (e.g., [63]), could be included in future work.

Algorithm Performance: To evaluate the performance of our optimization scheme we measured the time necessary to generate different trajectories. The runtime of the algorithm depends on the flight duration, the number of keyframes and the constraints which are incorporated into the optimization problem. Typical run times (Intel Core i7 4GHz CPU, Matlab's quadprog solver) are $\sim$1 sec for pure QPs (e.g., the trajectory in Fig. 3.8 had a flight time of 30 sec and was generated in 1.4 sec) and tens of sec for IQPs (e.g., the trajectory in Fig. 3.7 had a flight time of 20 sec and was generated in 14 sec). Optimizing over a receding horizon which is shifted along the

trajectory may be a fruitful strategy to speed up the algorithm. Another idea would be to split a trajectory in overlapping and reasonable constrained sub-trajectories and optimize them separately. Both approaches would negate the global optimality property of generated trajectories, requiring evaluation of real-world feasibility.

Onboard Control and Hardware: Once we generate trajectory control inputs these can be transmitted to a real quadrotor. Our real-time control system builds on the PIXHAWK autopilot software [64]. Desired positions along the motion plan, camera look-at vectors and target trajectories are transmitted from a ground-station via the Robot Operating System (ROS). An LQR (Linear-quadratic regulator) running on a dedicated single-board computer computes the necessary forces and moments to track the motion plan. These forces are then translated into low-level rotor and gimbal speeds by further controllers running on a PX4 FMU. We created result figures using two different quadrotor platforms: the 3DR Solo and a custom-build Pixhawk-based platform.

Design Tool: The 3D trajectory design tool has been implemented as Unity 3D tool which allows for easy adaptation and integration of a variety of IO devices. A further advantage of this design decision is that it is easy to develop augmented reality applications such as mixing real and virtual quadrotors in a racing scenario. We have interfaced the design tool with our optimization algorithm implemented using the Matlab optimization toolbox.

3.7 Results and Application Scenarios

Despite having used camera planning as running example we note that our method is general and can be applied in many different application scenarios. In particular, the discrete nature of the proposed IQP scheme makes it straightforward to incorporate application specific constraints. In this section we want to illustrate a number of interactive usage scenarios which we have implemented using our method.

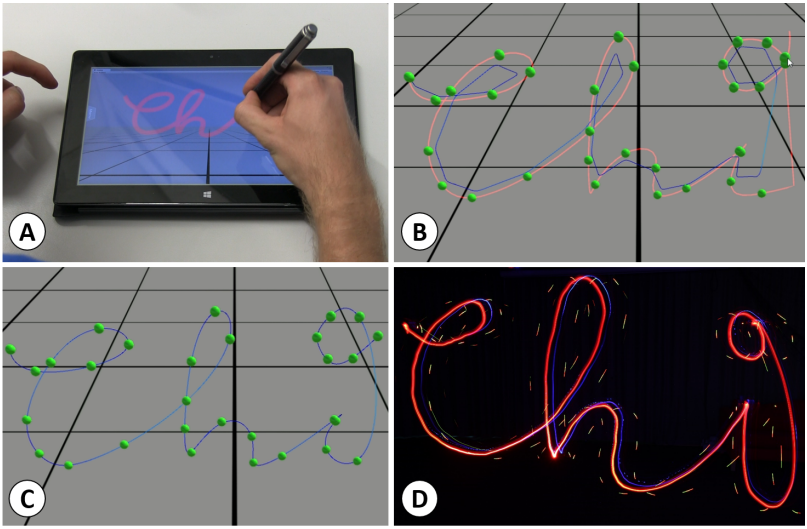

Figure 3.8: (A) Handwritten input. (B) Initial feasible trajectory can be overly smooth. (C) After iteration a feasible and visually pleasing trajecory is found. (D) Final result flown by MAV and captured via long-exposure photography.

3.7.1 Light Painting

Quadrotors have already been used in entertainment settings, in particular to create spectacular aerial light shows (cf. [37]). However, creating such complex and coordinated flight patterns is not possible with consumer grade technologies and hence has not been accessible to the end-user. Our tool allows for straightforward end-user design of such creative scenarios.

One such example is illustrated in Fig. 3.8. Here the user provides input position constraints by writing or sketching the desired shape. Our method then generates a feasible trajectory which as a side-effect

of minimizing snap also smooths the input strokes. However, the generated trajectory may not coincide with the desired output e.g. because it linearly interpolates the keyframes so that handwriting may not be legible anymore (see Fig. 3.8, B). The user can correct for this by changing the parameters of the optimization scheme (e.g., weights of the energies) or by adjusting keyframe positions and timings.

Once satisfied the trajectory can be flown by a real robot. In Fig. 3.8 we have mounted a bright LED to the robot and captured the flight path via long-exposure photography.

3.7.2 Racing

Another interesting application domain is that of aerial racing. First person drone racing is an emerging sport that requires a lot of expertise in manual quadrotor control. Our tool can bring this within reach of the end-user. As a proof of concept we have developed a simple aerial racing game. In this scenario a user can design a free-form race course, specifying length, curvature and other parameters as well as overall race-time.

For the race itself we implemented a semi-manual flight mode for which we changed the position controller, by remapping the feedforward term ($m\ddot{\mathbf{r}}_d$) of Eq. (3.14) to the joystick of a game controller. Thereby, the user can choose the direction and the strength of the feedforward-force allowing him to deviate with the quadrotor from the generated reference trajectory. Users can then, for instance, take a short cut in a curve or fly the trajectory with a higher velocity than generated by the optimization method. The score is calculated as a function of the deviation from the generated trajectory and the time needed to complete all laps. In other words, the player who managed to stay on the trajectory as fast as possible will win. We note that by manipulating the underlying controller, it would be possible to introduce further video game concepts such as player strength balancing into real-world quad racing. For example, allowing a player to temporarily race on a faster reference trajectory than his opponents.

3.7.3 Aerial Videography

Our main results stem from the application scenario of aerial videography. We have already mentioned the technical details and how we incorporate cinematographic goals into our optimization scheme. Here we briefly summarize a number of interesting and challenging video-shots (best viewed in video).

Fig. 3.7 illustrates a shot where a quadrotor flies over a toy castle and at the same time records it. Here the gimbal has to smoothly track the target just as the quadrotor swoops over the object and turns around its own axis once reaching the highest point. Such a shot composition is difficult to achieve manually due to the complicated quadrotor-camera-target coordination.

Even with conventional cameras, composition of multi target shots is a very challenging task. Aerial-videography makes this even more difficult due to the many degrees of freedom and complex geometric dependencies requiring coordination for smooth, jerk-free transitions from one to the next target while airborne. In Fig. 3.10 we illustrate a sliding shot, transitioning between targets – the two actors – while the camera is moving from left to right and steadily rising in altitude. Throughout the entire trajectory the oerientations of quad and camera never remain constant, yet the camera targets are kept in focus and the transitions are smooth. Flying such a trajectory manually would only be possible with two operators, one for steering the camera, the other the quadrotor.

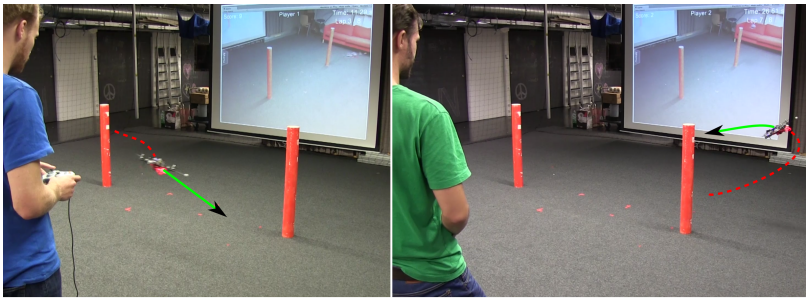

Figure 3.9: Two player aerial racing. User input is weighted with automatic control to adjust difficulty.

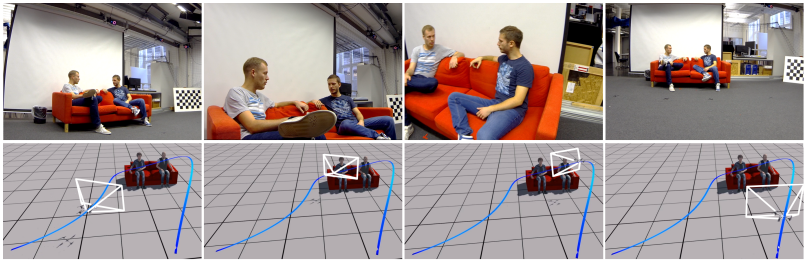

Figure 3.10: Multi target shot. Top row: frames of the video sequence shot by the onboard camera. Bottom row: according quadrotor positions shown in the preview of the design tool.

3.8 Technical details

In the work proposed here we use an approximation of a full, nonlinear quadcopter model for the optimization-based generation of trajectories. However, the resulting trajectories need to be flown by a real quadcopter and hence one must relate the approximate model to the full model of the quadcopter. Here we briefly summarize the modeling and control aspects necessary for replication of our method. A full introduction to this topic is beyond the scope of this work and we refer the interested reader to [61].

3.8.1 Quadrotor Model

A quadrotor is a robot with four identical rotors which generate a thrust and a moment orthogonal to the square they span. Our quadrotor model closely follows [59]. To describe the configuration of a quadrotor we define its position as the location of the center of mass in an inertial world coordinate frame ($\mathbf{x}_W$, $\mathbf{y}_W$, $\mathbf{z}_W$), and its attitude as the rotation of the body-fixed frame ($\mathbf{x}_B$, $\mathbf{y}_B$, $\mathbf{z}_B$) with respect to the world frame (see Fig. 3.11). The rotation matrix from body to world frame is then given by $R_{BW} = [\mathbf{x}_B \ \mathbf{y}_B \ \mathbf{z}_B] \in \mathbb{SO}(3)$. Each rotor of the drone has an angular speed ω_i and produces a force F_i and moment M_i, according to

$$F_i = k_F \omega_i^2, \quad M_i = k_M \omega_i^2,$$

where k_F and k_M are constants specific to the rotors. Therefore, the control input to the quadrotor can be written as $\mathbf{u}$ where u_1 is the net force in $\mathbf{z}_B$ direction and u_2, u_3, u_4 are the moments in $\mathbf{x}_B$, $\mathbf{y}_B$, $\mathbf{z}_B$ direction acting on the quadrotor. The input can be expressed in

terms of the rotor speeds ω_1, ω_2, ω_3, ω_4:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}, \tag{3.11}$$

where L is the distance from the axis of rotation of the rotors to the center of mass of the quadrotor.

The position of the quadrotor in the world frame can be specified according to Newton's equation of motion governing the acceleration of a mass point:

$$m\ddot{\mathbf{r}} = u_1 \mathbf{z}_B + m\mathbf{g} \in \mathbb{R}^3, \tag{3.12}$$

where $\mathbf{r}$ is the position vector, $\mathbf{g} = [0, 0, -g]^T$ is the gravity vector pointing along the $-z$ axis of the world frame, g is the gravitational constant and m is the mass of the quadrotor.

The Euler rotation equations are

$$\mathbf{M}_B = I\dot{\omega}_{BW} + \omega_{BW} \times I\omega_{BW} \in \mathbb{R}^3, \tag{3.13}$$

where $\mathbf{M}_B = [u_2, u_3, u_4]^T = [\omega_x, \omega_y, \omega_z]^T$ is the moment vector acting on the quadrotor in the body frame, ω_{BW} is the angular velocity of the body frame in the world frame and I is the moment of inertia of the quadrotor in the body frame.

3.8.2 Quadrotor Control

As can be seen from the equations Eq. (3.11), Eq. (3.12) and Eq. (3.13), the quadrotor configuration has 6 degrees of freedom but only 4 actuators. Therefore it is an underactuated system and cannot follow arbitrary trajectories in the configuration space. However, Mellinger et al. show that the system is *flat* [65] with respect to the 4 *flat outputs* $[\mathbf{r}, \psi]^T$ and thus a quadrotor can follow trajectories in this space, given that the corresponding inputs are bounded to values that the

quadrotor can achieve [59]. This flat output space is the configuration of our approximate quadrotor model.

We use the linear controller from [63] to generate the corresponding inputs for the quadrotor. The desired thrust along the z-axis of the body frame is computed as

$$\mathbf{F}_d = -K(\mathbf{x} - \mathbf{x}_d) + m(g\mathbf{z}_w + \ddot{\mathbf{r}}_d), \tag{3.14}$$

where $x = [\mathbf{r}, \dot{\mathbf{r}}]^T$ is the actual and $\mathbf{x}_d$ the desired position and velocity of the quadrotor and $m(g\mathbf{z}_w + \ddot{\mathbf{r}}_d)$ the feedforward term which compensates for gravity and known accelerations. The state feedback matrix K is computed using a linear quadratic control strategy with integral action.

The desired force F_d already defines two degrees of freedom of the quadrotor attitude. Using the nonlinear control strategy on $SO(3)$ described in [66] we employ the desired yaw angle ψ_d to compute the desired attitude R_{BWd} of the quadrotor:

$$\begin{aligned}
\mathbf{z}_{Bd} &= \frac{\mathbf{F}_d}{||\mathbf{F}_d||} \\
\mathbf{y}_{Bd} &= \frac{\mathbf{z}_{Bd} \times [cos(\psi_d), sin(\psi_d), 0]^T}{||\mathbf{z}_{Bd} \times [cos(\psi_d), sin(\psi_d), 0]^T||} \\
\mathbf{x}_{Bd} &= \mathbf{y}_{Bd} \times \mathbf{z}_{Bd} \\
R_{BWd} &= [\mathbf{x}_{Bd}, \mathbf{y}_{Bd}, \mathbf{z}_{Bd}],
\end{aligned}$$

where $\mathbf{y}_{Bd}$ are the desired x- and y-axis of the body frame. To control the attitude we can now calculate the desired moment vector $\mathbf{M}_{Bd}$ in $\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B$ direction,

$$\begin{aligned}
\mathbf{e}_R &= \frac{1}{2} vee\left(R_d^T R_{BW} - R_{BW}^T R_d\right) \\
\mathbf{e}_\omega &= R_{BW}^{-1}\left(\omega_{BW} - \omega_{BWd}\right) \\
\mathbf{M}_{Bd} &= -K_R \mathbf{e}_R - K_\omega \mathbf{e}_\omega,
\end{aligned}$$

where R_{BW} is the actual rotation of the quadrotor, $\mathbf{e}_R$ is the rotation error, ω_{BW}, ω_{BWd} are the angular and desired angular velocity, $\mathbf{e}_\omega$ is

the angular velocity error and *vee* is the *vee map* from $so(3) \rightarrow \mathbb{R}^3$. From $\mathbf{F}_d$ and $\mathbf{M}_{Bd}$ we can calculate the input u and thereby the velocities of the rotors needed to reach the desired position and yaw angle:

$$\begin{aligned} u_1 &= \mathbf{F}_d \cdot \mathbf{z}_B \\ [u_2, u_3, u_4]^T &= \mathbf{M}_{Bd}, \end{aligned} \tag{3.15}$$

where Eq. (3.15) is the projection of the desired thrust $\mathbf{F}_d$ on the actual z-vector of the body frame $\mathbf{z}_B$. Finally, using Eq. (3.11) we can compute the angular velocities ω_i corresponding to the input u.

3.8.3 Validity of Approximate Quadrotor Model

Following the approach in [67], we assume that the rotational dynamics of a quadrotor are fast compared to its translational dynamics thus we can describe the behavior of the quadrotor by the thrust vector $\mathbf{u}_r$ and the moment u_ψ along the body-frame z-axis. In the following we will only refer to the norm u_r of the force vector $\mathbf{u}_r$

Let $F_{\max}$ be the maximum force and $M_{\max}$ be the maximum moment each motor can produce. Then the bound on the maximum possible thrust that the quadrotor can achieve (i.e. all motors full on) is

$$u_r \leq u_{r,\max} = 4F_{\max}$$

and the bound on the maximum possible moment (i.e. two motors rotating in same direction full on and the other two off) is

$$u_\psi \leq u_{\psi,\max} = 2M_{\max}.$$

Because the force and moment are coupled it is not possible to achieve full thrust $u_{r,\max}$ and full moment $u_{\psi,\max}$ at the same time.

Let us now assume a stricter bound on the maximum moment of the quadrotor:

$$u_\psi \leq u_{\psi,lim} = \beta u_{\psi,\max}$$

where $\beta \in [0, 1]$. If we want to be able to achieve a moment of $u_{\psi,lim}$ at all times we have to take into account that in the extreme case two motors will be limited to a force of $F_{lim} = (1 - \beta)F_{\max}$ and thus the bound on the thrust of the quadrotor is

$$u_r \leq u_{r,lim} = (2 + 2(1 - \beta))F_{\max} = \left(1 - \frac{\beta}{2}\right) u_{r,\max}.$$

For example, if $\beta = 0.2$, i.e. bounding the moment to 20% of the quadrotors maximum moment the quadrotor can still achieve 90% of its maximum thrust at all times. These limits still allow the agility of a quadrotor to be sufficient for many use cases.

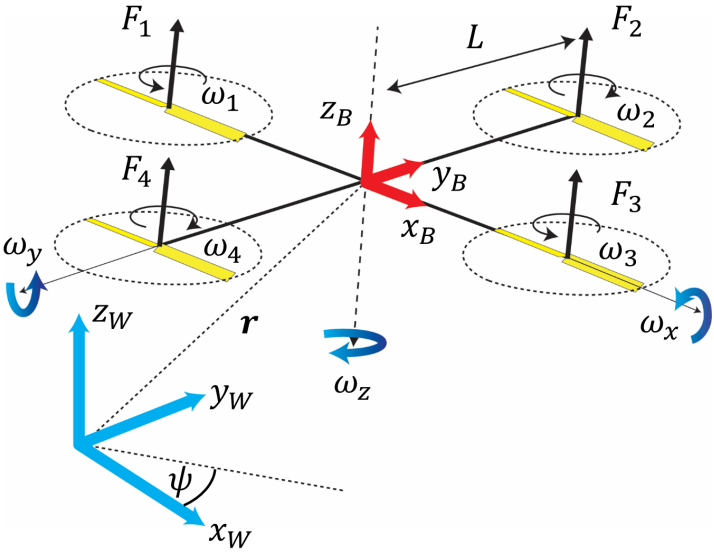

Figure 3.11: A quadrotor in 3D with its flat outputs (position $\mathbf{r}$, yaw angle ψ), world ($\mathbf{x}_W$, $\mathbf{y}_W$, $\mathbf{z}_W$) and body frame ($\mathbf{x}_B$, $\mathbf{y}_B$, $\mathbf{z}_B$), the rotational velocities of the quadrotor in each dimension (ω_x, ω_y, ω_z), the distance L from the axis of rotation of a rotor to the center of mass of the quadrotor, as well as the thrust forces F_i and angular velocities ω_i of each rotor.

3.9 Discussion

The optimization framework proposed in this chapter has proven to be powerful and versatile however there are of course a number of limitations. First, our goal is to enable non-expert users to design arbitrary MAV use cases. While the method is generic and designed to be extensible it does require expertise and effort to formalize further objectives (that we have not treated so far) and to integrate them into the algorithm. We believe that our high-level design tool bridges the gap between the underlying optimization algorithm and end-user goals sufficiently well. Nonetheless it is an interesting future research question how end-users could extend not only the use-cases we have demonstrated but also the optimization itself.

Currently all our application scenarios depend on a high precision indoor tracking system. This is a limiting factor as one would of course like to fly many of the examples outdoors using GPS sensing. To this end our method is generic and could be made to work with any localization system, in particular with GPS position data in outdoor scenarios. However, we have not implemented this and of course the localization accuracy would impact the exact results.

In summary we have proposed a user in the loop design tool for the creation of aerial robotic behavior. At its core lies an optimization-based algorithm that integrates low-level quadrotor control constraints and high-level human objectives. Therefore, we used a linear approximation of the quadrotor model enabling us to generate trajectories subject to the physical limits of a quadrotor. Stating the problem as discrete, additionally permits the easy incorporation of high-level constraints to support the user, for instance, in the creation of pleasing aerial footage. This allows users to concentrate on the creative and aesthetic aspects of the task at hand and requires little to no expertise in quadrotor control or the target domain. We have demonstrated the flexibility and utility of our approach in three different use cases including aerial videography, light painting and racing.

Chapter 4

Trajectory Planning for Multi-View Stereo Reconstruction

In chapter 3 we described a system that allows generation of flight trajectories for quadrotors with specific applications such as filming. While this system can produce trajectories for many use cases it needs a description of the intended behavior in the form of a sequence of desired quadrotor configurations or desired camera views during the flight. Generating such a keyframe description can be a difficult task in itself. Here we look at the problem of producing such a sequence of desired quadrotor configurations for producing 3D reconstructions in urban, crowded scenes. This chapter is based upon our work in [68].

We introduce a new method that efficiently computes a set of viewpoints and trajectories for high-quality 3D reconstructions in outdoor environments. Our goal is to automatically explore an unknown area, and obtain a complete 3D scan of a region of interest (e.g., a large building). Images from a commodity RGB camera, mounted on an autonomously navigated quadcopter, are fed into a multi-view stereo

reconstruction pipeline that produces high-quality results but is computationally expensive. In this setting, the scanning result is constrained by the restricted flight time of quadcopters. To this end, we introduce a novel optimization strategy that respects these constraints by maximizing the information gain from sparsely-sampled view points while limiting the total travel distance of the quadcopter. At the core of our method lies a hierarchical volumetric representation that allows the algorithm to distinguish between unknown, free, and occupied space. Furthermore, our information gain based formulation leverages this representation to handle occlusions in an efficient manner. In addition to the surface geometry, we utilize the free-space information to avoid obstacles and determine collision-free flight paths. Our tool can be used to specify the region of interest and to plan trajectories. We demonstrate our method by obtaining a number of compelling 3D reconstructions, and provide a thorough quantitative evaluation showing improvement over previous state-of-the-art and regular patterns.

4.1 Introduction

High-quality 3D reconstructions lie at the heart of many applications in computer graphics, AR/VR, robotics and GIS, architectural and urban planning. Motivated by this need for high-quality 3D models, techniques for the acquisition of building-scale geometry have rapidly advanced. Even with monocular cameras, 3D reconstructions of impressive quality can be attained using state-of-the-art multi-view stereo (MVS) methods [69, 70, 13, 71, 14, 18, 72, 73]. However, the final reconstruction quality depends to a large degree on the availability and *quality* of the set of input images [74, 75] (more is not always better).

Given the emergence of small and affordable aerial robots (MAVs), equipped with high resolution cameras, it is a natural choice to leverage these for image acquisition. In fact several commercial applications exist for this task (e.g., Pix4D [76] or Agisoft PhotoScan [77]).

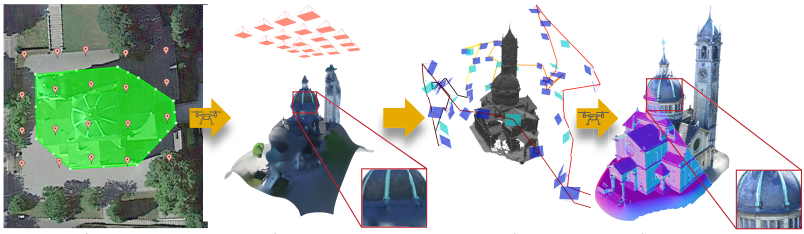

Figure 4.1: Depicted is our pipeline for 3D reconstruction of building-scale scenes with commercially available quadrotors. (A) A user defines the region of interest (green) on a map-based interface and specifies a pattern of viewpoints (orange), flown at a safe altitude. (B) The pattern is traversed and the captured images are processed resulting in an initial reconstruction and occupancy map. (C) We compute a viewpoint path that observes as much of the unknown space as possible adhering to characteristics of a purposeful designed camera model. The viewpoint path is only allowed to pass through known free space and thus the trajectory can be executed fully autonomously. (D) The newly captured images are processed to attain the final high-quality reconstruction of the region of interest. The method is capable of capturing concave areas and fine geometric detail.

Due to safety and absence of detailed environmental information (e.g., obstacles) such tools revert to flying regular grid-like patterns or circles at a safe overhead distance. However, the resulting viewpoints are in many cases insufficient for high quality 3D reconstruction: parts of the building such as walls and convex areas underneath overhangs may be occluded from overhead and images captured from far away may lack information important for reconstruction of fine details. Furthermore, such acquisition strategies do not directly account for complete coverage of the scene and do not explicitly reason about the choice of viewpoints with respect to expected feature matching quality and baseline between viewpoints. Moreover, current MAVs are battery constrained to 10-15 minute flight time, making intelligent viewpoint selection an even more pressing issue.

In this chapter, we propose an algorithm for the automated planning of viewpoint tours for aerial 3D scanning. We demonstrate its utility in the context of building-scale dense 3D reconstructions. We pose this problem in a mathematical optimization framework, based on the objective of *maximizing information* in terms of *observed space* weighted by uncertainty from as few viewpoints as possible. This objective results in *minimizing unobserved space*. Starting from a coarse input scan the method selects a small number of viewpoints, each providing additional information about the environment, and considers constraints in terms of stereo matching, flight time of the MAV and collision freedom. The resulting plan can be flown by a real quadcopter and the images are processed via a SfM & MVS pipeline [13, 14, 18] to attain high-quality 3D models.

The above optimization problem involves instances of the coverage set problem for viewpoint selection and the traveling salesman problem for path planning. Both problems are known to be NP-hard [78]. In order to make the problem computationally tractable, we introduce an approximated camera model to measure the expected information gain (IG) contribution from an individual viewpoint, independently from all other viewpoints. While this camera model does not accurately represent monocular viewpoints in MVS settings, it allows us to formulate the problem with a submodular objective function [1], mak-

ing the computation feasible while giving good approximation bounds. We show that both model and problem approximation yield very good results in practice, and can be effectively combined with an edge selection strategy during trajectory generation that favors camera motion which will lead to good sparse matches and hence good camera pose estimates.

We have developed a simple pipeline that is depicted in Fig. 4.1. This pipeline allows novice users to reconstruct building-scale outdoor scenes with little prior knowledge and effort. Based on a user-specified region of interest and no-fly zones a simple overhead pattern is flown for bootstrapping. An initial reconstruction from the collected images is used for planning a sequence of viewpoints through free space and within the maximum travel budget. Finally, the trajectory is flown and a dense 3D model is attained via MVS reconstruction. We have produced a number of reconstructions including a free-standing office building, a historical downtown building in cluttered environment, and an entire renaissance church . Furthermore, we evaluate our method quantitatively using a 3D game engine and show that it outperforms both regular patterns and the previous state-of-the art [80].

In summary, we contribute: i) an optimization criteria that is based on volumetrically-represented information content and that maximizes information about uncertain space subject to a travel budget and free-space constraints; ii) a camera-model that makes the optimization problem suitable for a sub-modular formulation, iii) a recursive strategy to approximately solve the sub-modular optimization problem, iv) a quantitative comparison with strong baselines on synthetic scenes and qualitative results on three varied and challenging outdoor scenes, v) an open-source user interface (see Fig. 4.2) that allows visualization of the scanned occupancy maps and 3D reconstructions and handling of the viewpoint graph and path planning [79].

Figure 4.2: Shown is the user interface for our method. The interface allows the visualization of the region of interest (yellow lines), the occupancy map (grey boxes), the 3D reconstruction in form of a Poisson mesh or point cloud. For inspection and debugging purposes direct voxel selection via mouse or raycasting is supported as well. Furthermore the interface allows to control viewpoint generation, motion path computation and the computation of viewpoint paths. The user interface is available as open source code [79].

4.2 Related work

Our work builds upon a large body of work in the computer graphics and robotics literature. Here we review the most salient work in aerial 3D scanning, path planning and monocular 3D reconstruction.

3D Reconstruction: SfM and MVS The theory behind structure from motion (SfM) and multi-view stereo reconstruction (MVS) methods [81, 11] forms the foundation for an impressive diversity of image-based 3D reconstruction approaches. SfM methods can now obtain city-scale reconstructions from unstructured image collections [82, 83, 84, 85, 86] with remarkable levels of quality [87, 75, 88, 15, 89]. Many of these algorithms are now publicly available as open source projects. For example, CMPMVS [69], MVE [13], SMVS [71], COLMAP [14, 18], or as commercial solutions, such as Pix4D [76] or Agisoft Photoscan [77]. A very recent work by Knappitsch et al. [73] provides an overview of state-of-the-art SfM and MVS methods, and introduces an impressive benchmark dataset along with an evaluation for such approaches. We incorporate domain knowledge about the SfM and MVS process into our planning algorithm but our method is agnostic to the exact underlying SfM and MVS implementation.

Image Selection for MVS It has been shown that the quality and speed of most MVS algorithms significantly depends on the selection of input images and not all input images contribute equally to reconstruction quality, leading to unnecessary processing time if all images are used. Using too many images can even lead to degraded quality in the reconstruction [74]. Most MVS pipelines use relatively simple techniques to select images such as k-nearest images. Several automatic techniques for image selection have been proposed based on heuristics [90] or on contours [91]. More recently, Hornung et al. [21] proposed a method that incrementally selects images based on maximizing coverage of a proxy geometry. It has also been proposed to leverage viewpoint entropy (similar to IG in active vision) for view

selection in image-based rendering [92]. While not directly focusing on MVS an image selection process based on covariance propagation has been proposed for incremental SfM [93]. However, all these techniques either expect a complete set of already captured input images or update the 3D reconstruction on-the-fly, making them unsuitable for our settings of finding the best viewpoints for dense aerial 3D reconstruction.

RBG-D Reconstruction In addition to monocular methods, many 3D reconstruction approaches are inspired from range sensing technology. A fundamental component is the volumetric fusion work by Curless and Levoy [94] whose implicit surface representation is at the core of most prominent real-time RGB-D reconstruction methods, such as Kinect Fusion [95] and many others [96, 97, 98]. Poisson Surface reconstruction is another widely used 3D reconstruction method, where the 3D surface is defined by an energy minimization problem over an implicit, volumetric function [99, 10].

Aerial 3D reconstruction Several commercial tools aid users in flight planning and processing of aerial imagery for 3D reconstruction [76, 100]. However, these tools only produce regular overhead patterns that have to be flown at a safe obstacle-free altitude. Due to a lack of any prior model information these tools are also incapable of reasoning about coverage and viewpoint quality, limiting the resulting 3D reconstruction completeness and quality. A number of systems leverage MAV-mounted RGB-D sensors to create 3D reconstructions in real-time [101, 102, 103, 104]. However, these focus on the reconstruction method and do not consider the problem of planning trajectories for reconstruction.

Exploration and active vision Planning trajectories through (partially) unknown scenes is a classic problem in the robotics literature (see [105] for a recent review). Frontier-based exploration algorithms [106] are often used for autonomous mapping and exploration of the

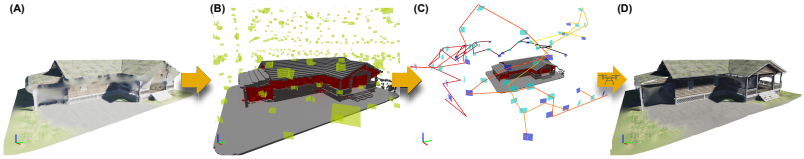

Figure 4.3: Overview: (A) The method is initialized via images from a regular overhead pattern (not shown). (B) We generate a set of viewpoint candidates (yellow) and use a submodular optimization formulation to find an optimized viewpoint path maximizing information about uncertain space (voxels in red). (C) The planner generates a free-space trajectory and takes constraints on SfM & MVS reconstruction such as preference for fronto-parallel views and matching quality into consideration. Where necessary viewpoints for sparse matching (cyan) are inserted. (D) The final, high quality reconstruction is attained via a SfM & MVS pipeline [13, 14, 18].

environment using stereo [107], RGB-D, or monocular cameras [108]. The goal in our work differs since we are not only interested in a coarse map for navigation but in a dense, high-quality 3D scan of building-scale scenes.

Scanning an a priori unknown scene is a challenging problem due to the lack of available information and absence of useful priors. An interesting approach to guide the recording process was proposed in [109] where an online SfM system is combined with measures for redundancy and sampling distance to give live feedback on the expected reconstruction quality to a user. However, in our context we are already given an initial coarse model of the scene and reason about the whole recording sequence in volumetric space before executing it. In active vision approaches, planning of the next view is done on-the-fly by incorporating information from previous scans, an approach known as next-best-view (NBV) and itself an instance of the set cov-

erage problem which is NP-hard. Much work has been dedicated to efficient approximations [78] which have been utilized for 3D scanning [110, 111, 112, 113]. Similar to our formulation such approaches leverage information gain (IG) as optimization quantity. Another interesting approach uses two independent cameras that can collaborate to form a stereo pair [114]. In contrast to our work these approaches either aim for establishing a coarse map for navigation or have not shown high quality dense reconstruction resulting from planned viewpoints.

In graphics, several approaches for quality driven and automated 3D scanning have been proposed including NBV approaches using volumetric [115] and poisson mesh based [116] metrics for scanning quality. Fan et al. [117] propose view- and path-planning algorithms to scan collections of small objects using a structured light scanner. Xu et al. [118] propose a method for automatic scanning of indoor scenes but focus on guiding the scanning process via object recognition and scene understanding. While these works share our goal of optimizing for reconstruction quality, all of them are designed for structured light scanners, robotic arms, or wheeled robots and comparatively small scenes and hence are not applicable in the context of aerial, monocular (offline) scanning.

In photogrammetry, the task of finding the best viewpoints for accurate 3D scanning is known as the photogrammetric network design problem (NDP). Early work [119] already highlights the difficulties of the problem including high non-linearity and multi-modality which makes it difficult to model in typical optimization frameworks. Model-based approaches [120] and such driven by model uncertainty [121] have been proposed but only deal with the geometric aspect of the task in highly-controlled environments and idealized localization settings (i.e., using fiducial markers) but do not consider the influence of texture and appearance on SfM and MVS pipelines.

The methods in Hoppe et al. [122] and Bircher et al. [123] use a prior mesh to reason about coverage of the scene; however, they only use the mesh for collision checking which might be insufficient for non-watertight meshes with unobserved parts. Their main goal is to

find viewpoints that cover the whole scene, whereas we maximize the observed information about a specific object of interest. Once budget constraints like the number of viewpoints or flight-time are introduced such approaches may lead to arbitrarily bad viewpoint selections. The work in [124] performs reconstruction online by iteratively capturing a planned sequence of next-best-views and computing depth maps with a fast Multi-View-Stereo method. In contrast our focus is not on planning in an online fashion but on reconstructing a high quality surface model at building scale. The method in [80] is most similar to ours in spirit. The authors also formulate the viewpoint selection as a submodular optimization problem. However, to solve the problem the authors perform two problem approximations resulting in a mixed-integer linear program that is solved with an off-the-shelf solver. Due to these approximations it is not clear whether any approximation bound can be given for the original problem. More importantly, in all our experiments the solver never finds the optimum and it is up to the user to decide how long to run the optimization. For reasonable runtimes of 10 minutes we observe that our approach consistently outperforms this method.

To the best of our knowledge, the view selection and path optimization problem for dense aerial reconstruction has not been formulated in the same way as proposed here and has not been solved in the same efficient manner as we do.

4.3 System Overview

The aim of our method is to automate high-quality 3D scanning of building-scale scenes using commodity quadrotors equipped with a single RGB camera. To this end, we have developed a simple pipeline, illustrated in Fig. 4.1 and 4.3, which is built on a mathematical optimization formulation for viewpoint and trajectory planning. In the following, we provide an overview of the workflow from a user's perspective and then detail the planning algorithm.

First, a user defines a region of interest (ROI) and specifies a simple

and safe overhead pattern via a map-based interface to acquire an initial set of images. In urban environments, it is also possible to specify no-fly zones in order to avoid other buildings, powerlines, and to adhere to airspace regulations. The quadrotor flies this regular pattern and records an initial set of images. These recordings are then processed via a state-of-the-art SfM and MVS pipeline by Schönberger et al. [14, 18] to attain camera poses together with depth and normal maps for each viewpoint. To generate a 3D surface reconstruction, the depth maps are fused into a dense point cloud, and utilizing the Poisson Surface Reconstruction method [10] a mesh is extracted (Fig. 4.3, A).

It is important to note that this initial reconstruction is highly inaccurate and incomplete since the viewpoints stem from a simple, regular pattern, flown at relatively high altitude to avoid collisions.

In addition to the initial reconstruction, we compute a volumetric occupancy map containing occupied, free-space, and unobserved voxels. Each voxel also carries with it a measure of observation quality. We describe this occupancy map in more detail at the end of this section. Our occupancy map is based on the implementation from [9].

The occupancy map (Fig. 4.3, B) is used during planning to reason about free-space and collision freedom as well as approximation of the observable surface area from any given viewpoint and the (remaining) uncertainty about the scene. The main objective of our optimization formulation is to maximize total information (i.e. certainty about voxels in the region of interest) while staying within the travel budget of the quadrotor and respecting constraints imposed by SfM and MVS. Intuitively, the total information corresponds to the observation count of all voxels within the region of interest. In theory, given lambertian surfaces, once all voxels have been observed multiple times (from different non-grazing angles) the entire surface can be reconstructed with high quality. This goal has to be traded-off with limited battery time of the robot and computational cost as the evaluation of all possible viewpoints during planning is clearly infeasible.

Furthermore, it has been shown that at some point adding views yields diminishing returns [125, 74, 21]. Therefore, we propose an

efficient way to generate and evaluate viewpoint candidates (see Fig. 4.3, B) during planning, alongside a method to find a sequence of 'good' viewpoints to be flown in order to maximize reconstruction quality.

Fig. 4.3, C shows the output of our planning method, where viewpoints that were added due to their contributed information are rendered in blue. Additional viewpoints that were added to ensure that the SfM & MVS backend can register all images into a single reconstruction are rendered in cyan. The edges are color-coded to signal MAV progress along the path. The plan is then executed by the drone and the acquired images are used to update the 3D model (Fig. 4.3, D). Note how convex areas underneath the front-porch and garage roofs have been carved and how overall detail has been increased. However, there are still parts of the scene that have not been reconstructed properly. This is due to approximations simplifying assumptions in our formulation. We discuss these limitations in 4.6.

Volumetric Occupancy Map The occupancy map OM is essential in distinguishing between occupied, free and unobserved space. This is encoded by an occupancy value $oc(\tau) \in [0, 1]$ for each voxel $\tau \in OM$. Here we overload the term occupancy to encompass both a known occupancy and an unknown occupancy, i.e. a value close to 0 encodes a known empty voxel, a value close to 1 encodes a known occupied voxel and a value close to 0.5 encodes an unknown voxel.

We initialize the occupancy map with occupancies of $\sigma = 0.5$ (i.e. unknown). From each viewpoint, we cast a ray through the center of each pixel until we reach the 3D point given by the depth value $[94, 9]$. The occupancy of each traversed voxel is updated according to an inverse sensor model (see Sec.4.7 for more details). The size of a voxel is $vs = 0.2m$ in all our experiments. We denote the position of a voxel τ as $\tau.\mathbf{p}$. In the following we only consider voxels inside the region of interest when computing the viewpoint information. For illustration purposes we show an example of an occupancy map together with the corresponding dense point cloud in Fig. 4.4.

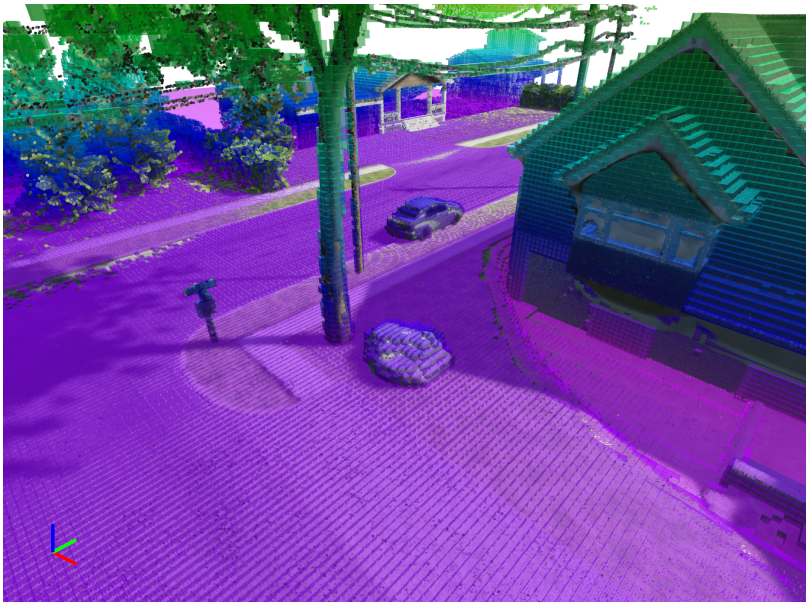

Figure 4.4: Visualization of an occupancy map. The occupancy map is rendered with a jet colormap by the height of the voxels. Here we render transparent voxels to illustrate the alignment with the corresponding dense point cloud.

Number of Images for 3D Reconstruction As discussed in the related work section it it not always beneficial for the final 3D reconstruction to include additional images. In fact a very high number of images will often lead to a 3D reconstruction of lower quality than one from a moderate number of images.

Such detrimental effects can be explained by first realizing that Multi-View Stereo is an inherently ambiguous process because dense correspondences between pixels from different images are established based on appearance alone. These correspondences will inevitably contain errors. While such errors are reduced by cross checking or regularization of depth and normal maps they cannot be fully eliminated. When including more and more images the number of such errors will increase. At some point more images will provide no more or very little additional surface information while the number of errors keeps increasing. Thus an increasing number of images can have detrimental effects on the final reconstruction. We demonstrate this experimentally on the reconstruction of a building facade as shown in Fig. 4.5. We recorded a video of a building facade and retrieved sets of 100, 200 and 500 images by extracting frames uniformly in time. In Fig. 4.5 we can see that the reconstructions become more noisy with more images. Note that these experiments were performed with Colmap [14, 18].

These results demonstrate that it is not enough to simply take an excessive amount of images of a scene but instead a moderate number of *good* images should be captured. Our system implicitly achieves this goal due to our saturating coverage function as detailed in the next Section.

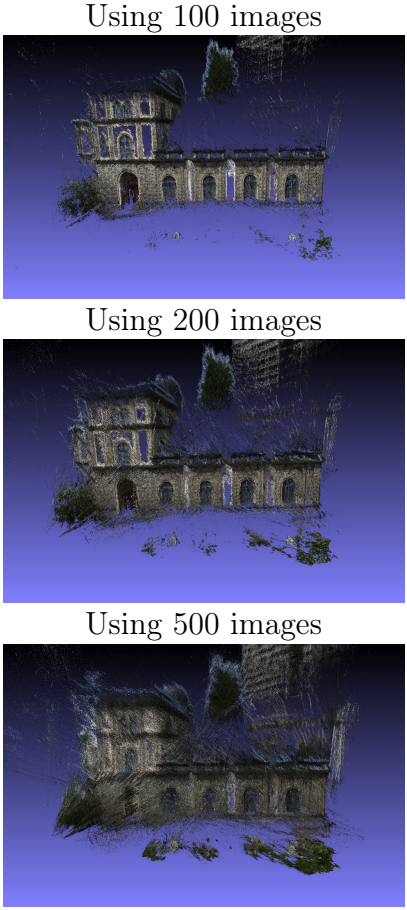

Figure 4.5: Quality comparison of 3D reconstruction with different numbers of images. Shown are 3D reconstruction of a building facade from 100, 200 and 500 images. We see that already 100 images result in a good reconstruction and an increasing number of images leads to detrimental effects on the reconstruction quality. Note that the images were extracted uniformly in time from the same video sequence.

4.4 Method

In this section, we discuss our optimization framework to obtain viewpoint trajectories for 3D scanning. One difficulty that presents itself is that the final objective of 3D reconstruction quality cannot be measured directly due to the absence of ground-truth data and the offline nature of the SfM and MVS pipeline. Hence, we require a surrogate quantity that can be used for optimization. At the core of our method lies an objective function based on the total information gained by the collection of viewpoints along a computed trajectory. We first introduce the formal optimization problem and then discuss how an approximate camera model can be leveraged to modify the original problem to allow for efficient maximization.

4.4.1 Optimizing viewpoint trajectories

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

We assume that we are given a graph $G = (C, M)$ of viewpoint candidates C alongside observed voxels and motions M between viewpoints as edges. Each viewpoint $v \in C$ has an associated position and orientation denoted as $v.\mathbf{p}$ and $v.\mathbf{q}$. We require each of the motions in M to be collision-free and that the connected viewpoints can be matched later on in the MVS pipeline. We detail the construction of this graph in Sec. 4.4.4.

The goal of the method is to generate a trajectory (i.e. a path through a subset of the nodes in the candidate graph) for the quadcopter that yields good reconstruction quality and fulfills robot constraints . In our case we are given a maximum travel distance of L_{max}. Let $VP = (vp_1, \ldots, vp_n)$ be the sequence of viewpoints to be traversed during image capture where $vp_i \in C$. We denote with $\mathrm{L}(VP)$ the geometric length of the trajectory VP and with S the set of all sequences VP.

Formally, we want to solve the following optimization problem:

$$VP^* = \underset{VP \in S}{\operatorname{argmax}} \; I(VP) \qquad (4.1)$$
$$\text{such that } \mathrm{L}(VP) \leq L_{max} \quad ,$$

where $I(VP)$ is our objective function that measures the amount of information contributed by the respective viewpoints. This optimization problem is equivalent to Eq. (2.12) described in Section 2.4. We can write the objective function as a sum over the information of each non-free-space voxel

$$I(VP) = \sum_{\tau \in OM \setminus OM_{free}} VI(\tau, VP) \qquad (4.2)$$
$$OM_{free} = \{\tau \in OM : oc(\tau) \leq oc_{free}\} \quad , \qquad (4.3)$$

where $VI(\tau, VP)$ is our camera measurement model and specifies how much information of voxel τ is contributed by the traversed viewpoints VP (details in Sec. 4.4.2) and oc_{free} is a lower threshold that determines when a voxel is considered to be free space (see Appendix for details).

Solving Eq. Eq. (4.1) is in general prohibitively expensive for non-trivial real world problems [1] since the total information of a voxel depends on the set of all traversed viewpoints. As our viewpoint graphs contain $5,000 - 10,000$ viewpoints it is computationally infeasible to enumerate all these viewpoint sets to find the best one.

4.4.2 Submodular voxel information

To make the problem tractable, we approximate the contributed information of a voxel by assuming that a single viewpoint v can directly provide information about the 3D surface, i.e., for a single voxel τ and viewpoint v the contributed information can be written as $vi(\tau, v)$. We do this by incorporating terms that encourage close-up and fronto-parallel views. The computation of the contributed information is

detailed in Sec. 4.4.4 (see Eq. Eq. (4.7)) and depends on the incidence angle of the observation ray and the normal of the voxel.

Clearly, this model is far from reality since it entirely ignores the stereo-matching process and does not explicitly encourage the selection of images from diverse viewpoints. However, it does allow us to re-formulate our optimization problem in a way that it exploits submodularity in the objective function and therefore allows for more efficient maximization of the problem [1]. Note that to explicitly incorporate stereo matching into the objective function we would want a view to give a high incremental objective value if another view can be matched to it. Let $A, B \subset 2^S$, $A \subset B$ and $x \in S \setminus B$ where S is the set of all possible camera poses. Let us assume that B contains a view that allows good stereo matching with x while A contains no such view. To incorporate stereo matching into the objective function $I : 2^S \rightarrow \mathcal{R}$ we would like to have $I(B \cup x) - I(B) \gg I(A \cup x) - I(A)$. We see that such a function is not submodular as it would violate the submodularity condition $I(B \cup x) - I(B) \leq I(A \cup x) - I(A)$.

We can finally write the total information of a voxel τ contributed by a set of viewpoints as

$$VI(\tau, VP) = \min\left(1, \sum_{v \in VP} vi(\tau, v)\right) \quad , \tag{4.4}$$

where the *min* ensures that the total information for each voxel saturates at 1. It is easy to see that this objective function is submodular [1] by writing the information gain resulting from adding v to the viewpoint sequence VP:

$$\begin{aligned} IG(\tau, v, VP) &= VI(\tau, \{v\} \cup VP) - VI(\tau, VP) && (4.5) \\ &= \min\left(vi(\tau, v), 1 - VI(\tau, VP)\right) \quad . && (4.6) \end{aligned}$$

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

As mentioned above, our submodular objective function does not directly incorporate stereo matching. We would like to encourage the

optimization to select multiple views of the same voxel to account for the requirements of stereo matching. To this end we reduce the contributed information per viewpoint by a discount factor ξ. This encourages on average $1/\xi$ cameras to observe the same point, leading to good MVS reconstructions in practice. Please also note that our formulation continues to enforce stereo matching constraints via the edge selection strategy discussed in Sec. 4.4.4.

4.4.3 Maximizing the submodular formulation

Our camera model exposes desirable structure in the optimization problem Eq. Eq. (4.1). Since the individual viewpoints now provide diminishing returns given other additional viewpoints and more viewpoints can never reduce the total information, the objective function $I(VP)$ is both *monotone* and *submodular*. While the problem is still NP-complete in general, submodularity provides guarantees on the approximation quality of a greedy algorithm for the when we are allowed to select a fixed number of viewpoints [1]. This guarantee does not hold anymore when we introduce a travel/time budget constraint on the path through the selected viewpoints . By combining the greedy algorithm and the cost-benefit algorithm [29, 126] and choosing the better solution we are guaranteed to be within $(1 - 1/e)/2 \simeq 0.32$ of the optimal solution. However, the cost-benefit algorithm requires $|C|$ evaluations of the contributed information $vi(\tau, VP)$ for each added viewpoint and thus does not scale well with an increasing set of viewpoint candidates and an increasing travel/time budget. The algorithm proposed in [27] also solves the submodular optimization problem with a travel-budget constraint. The algorithm fixes a start and end viewpoint and follows a recursive strategy by selecting a *middle* viewpoint and splitting the trajectory into a first (start to middle) and second part (middle to end). The travel-budget is also split into a portion for the first and second part. The first and second part are then computed by taking a recursive step. Interestingly, the authors are able to provide an approximation ratio of $\log O*$ for this algorithm where $O*$ is the optimal value of the objective function. However, the

algorithm is severely restricted in practice as it enumerates all possible *middle* nodes and all possible splits of the travel-budget (which is assumed to be integer valued) for each recursion step. Also note that the objective and the contributed information of each viewpoint are assumed to be integer valued (otherwise we could just scale all information values to achieve a better approximation ratio). In our setting with real-valued budgets and travel costs and a high number of viewpoint candidates this algorithm is not feasible.

We introduce a practical adaptation of the method in [27] to solve Eq. Eq. (4.1). In all our experiments this method performs favorably and always achieves a better solution than the greedy method. Note however, that our solution is only an approximation as the problem is still NP-complete in general.

Our approach can be seen as striking a middle-ground between the greedy algorithm and the cost-benefit algorithm. When observing the greedy algorithm one usually sees that the next viewpoint being picked is very far away from already selected viewpoints. This makes sense as viewpoints far away will have little overlap with already selected viewpoints and there is no penalty for large distances between selected viewpoints. This often leads to suboptimal selections as much of the viewpoint budget is used up very early and later on viewpoints can only be selected very close to the travel paths between the earlier viewpoints. On the other hand, the cost-benefit algorithm usually ends up selecting viewpoints that are very close to existing viewpoints due to the penalty of choosing viewpoints with large distances to the already selected ones. This often results in clusters of viewpoints with low coverage. Our method will typically also pick new viewpoints that are far away from existing viewpoints as we do not directly penalize distances in the selection. However, after selecting a viewpoint we split the budget into a first part (before reaching that viewpoint) and a second part (after reaching that viewpoint). We then recursively continue selecting further viewpoints in the first part and only later continue with the second part.

Our method proceeds as follows: the budget is split into a first and second part and a middle node (viewpoint) is selected. A recursion

step is made for the first part and the second part. In contrast to [27] we only perform an equal split of the budget for the first and second part and select as the middle node the viewpoint with the highest information gain that is reachable with the current budget. The recursion naturally ends when no new viewpoint can be reached with the available budget. Additionally, we adjust the budget for the recursion of the second part to make use of all remaining budget after the recursion of the first part has finished. A formal description of the method is given in Alg. 2. In Fig. 4.6 we show an example of a viewpoint path computed with our method.

4.4.4 Viewpoint candidate graph

In this section we detail the generation of the candidate viewpoint graph including computation of contributed information per viewpoint and generation of free-space motion paths. The candidate viewpoint graph G consists of a large number of candidate viewpoints and the corresponding observed voxels as nodes and free space motions between viewpoints as edges. Note that we first generate the set of candidate viewpoints and afterwards compute motion paths between them.

Ideally, the set of candidate viewpoints should consist of all camera poses that are useful for reconstructing the surface in the region of interest. At the same time the set should be as small as possible to enable faster computation on the set. We note that a pair of images with a fixed baseline will be visually more similar if the images are taken far away from the region of interest. Thus the candidate generation process (see Alg. 3 in Appendix) is designed to perform a denser sampling of viewpoints inside or in the vicinity of the region of interest and a less dense sampling for viewpoints far away from the region of interest. The set of candidate viewpoints is seeded either by the manually specified viewpoints or those stemming from an earlier iteration of the planning algorithm. These initial candidates are also added to an exploration queue. To sample new 3D candidate positions we take the first 3D position from the exploration queue

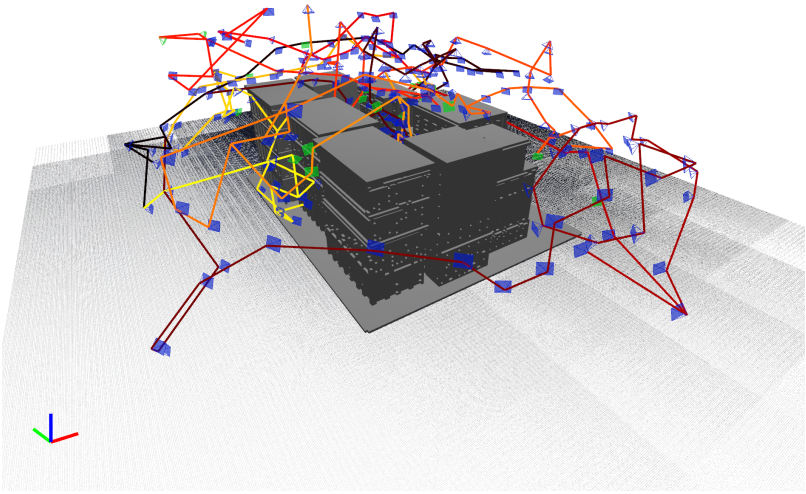

Figure 4.6: Visualization of a viewpoint path computed with our method. The viewpoints on the path are rendered as transparent blue camera frustums (green cameras were added to fulfil the sparse matching constraint as described in Sec. 4.4.4). The motion segments are rendered with a heat colormap by distance from the start viewpoint. Also shown is a dense point cloud of the scene within the region of interest (yellow lines).

and generate 6 new positions by adding an offset in the $-x$, $+x$, $-y$, $+y$, $-z$ and $+z$ direction respectively. The resulting positions are discarded if they are too close to existing viewpoint candidates or do not lie in free space, otherwise they are added to C and to the exploration queue. The orientation of new viewpoints is determined by random sampling with a bias towards the region of interest.

A viewpoint is considered to lie in free space if the occupancy of all voxels intersecting with the drone's bounding box is below the threshold oc_{free} (see Appendix for more details).

We furthermore require the camera orientation to fall within the limits of the physical camera (i.e., camera roll $\phi = 0$ and pitch $\theta \in [-\pi/2, 0]$). We run this procedure until no more new viewpoints can be added or we reach a maximum viewpoint budget. For typical scenes, $5,000$ - $10,000$ viewpoint candidates are generated (see Fig. 4.3, B).

Importantly G is not arranged in a uniform grid. Instead we let the sampling offset between neighboring viewpoints grow with increasing distance from the region of interest. This prevents an explosion in the number of viewpoints for larger scenes.

In Fig. 4.7 we visualize an example of a viewpoint graph. As can be seen the graph is not arranged in a uniform grid and has a higher density close to the region of interest. In Fig. 4.8 the corresponding motions of the viewpoint graph for a single viewpoint are shown.

Viewpoint information

After generating viewpoint candidates, we compute the visible voxels for each viewpoint needed to evaluate the corresponding contributed information by ray-casting into the occupancy map. Compared to a simple rendering of the coarse Poisson mesh from the initial scan the ray-cast ensures that we handle occlusions by objects that were not reconstructed (i.e. unknown voxels). Note that this operation is fairly expensive but only has to be performed once for each viewpoint candidate.

The information $vi(\tau, v)$ contributed by observing a voxel τ with

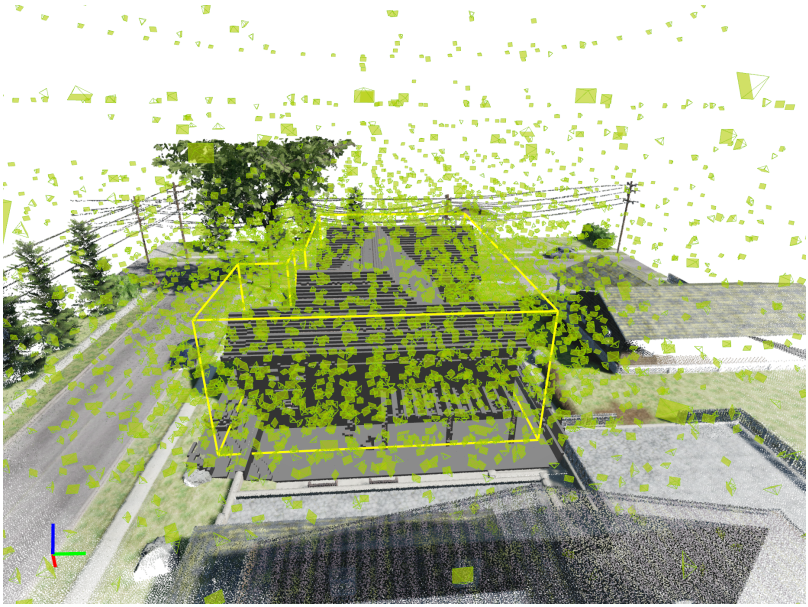

Figure 4.7: Visualization of a viewpoint graph with 6000 viewpoints. The viewpoints in the graph are rendered as transparent yellow camera frustum. The viewing direction of the viewpoints is biased towards the region of interest and the density of viewpoints is higher close to the region of interest. Note that for visualization purposes we only show the edges (i.e. motions) of a single vertex in the graph. Also shown is a dense point cloud of the scene within the region of interest (yellow lines).

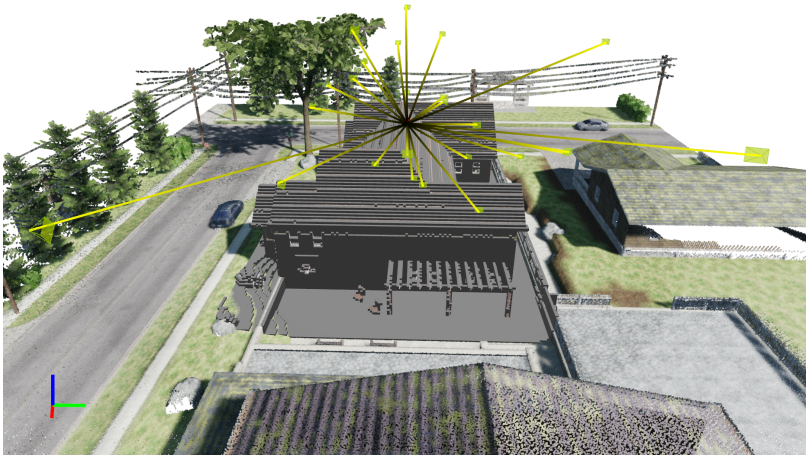

Figure 4.8: Visualization of motions in the viewpoint graph. The viewpoints in the graph are rendered as transparent yellow camera frustum. The motions are rendered as line segments connecting the viewpoints. Note that for visualization purposes we only show the edges (i.e. motions) of a single vertex in the graph. Also shown is a dense point cloud of the scene within the region of interest (yellow lines).

viewpoint vp is then given by

$$\begin{aligned} vi(\tau, v) &= \frac{1}{\xi} vi_i(\tau, v) vi_r(\tau, v) \\ vi_i(\tau, v) &= \exp(-\beta_i^F \max(\gamma - \beta_i^T, 0°)) \\ vi_r(\tau, v) &= \exp(-\beta_r^F \max(px(\tau, v) - \beta_r^T, 0px)) \quad , \end{aligned} \tag{4.7}$$

where $vi_i(\tau, v)$ and $vi_r(\tau, v)$ are incidence- and resolution-dependent factors respectively, $\gamma = \arccos((v.\mathbf{p} - \tau.\mathbf{p}) \cdot n(\tau, v))$ is the angle between the incident viewing ray and the surface normal (extracted from the mesh) and $px(\tau, v) = \frac{f\,vs}{d(\tau.\mathbf{p}, v.\mathbf{p})}$ is the number of pixels that a fronto-parallel voxel stretches if projected onto the viewpoint's image plane with a focal length f. Note that $d : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$ refers to the Euclidean distance. Both factors have the same functional form: a constant value of 1 up to a certain threshold β_i^T, β_r^T followed by an exponential decrease with falloff factors β_i^F, β_r^F. We empirically determined the following values to work well on a number of scenes: $\beta_i^T = 25°$, $\beta_i^F = \frac{1}{25°}$, $\beta_r^T = 6px$, $\beta_r^F = \frac{1}{3px}$. These values are used for all our experiments.

In Fig. 4.9 we visualize the ray-cast obtained from a camera viewpoint. Voxels that are hit by the ray-cast are colored from gray to red with gray indicating a low and red indicating a high value of $vi(\tau, v)$.

Note that we compute the normal of a voxel in a per-viewpoint fashion. This is accomplished by rendering the Poisson mesh of the initial reconstruction from the corresponding viewpoint for which we are computing the voxel information. Furthermore, we consider the voxel's projected size to account for the physical camera resolution. Clearly, the voxel information vi depends on the quality of the initial Poisson mesh. However, as our incidence factor vi_i shows a flat response around an incidence angle of 0° (i.e. looking parallel to the surface normal) the voxel information vi is only slightly effected by small amount of noise in the normals of the Poisson mesh. To handle normals that are close to creases of the surface, we compare the distance of camera to mesh and camera to voxel. If these distances differ too much ($\geq 0.5m$) we assume an unknown normal and set the inci-

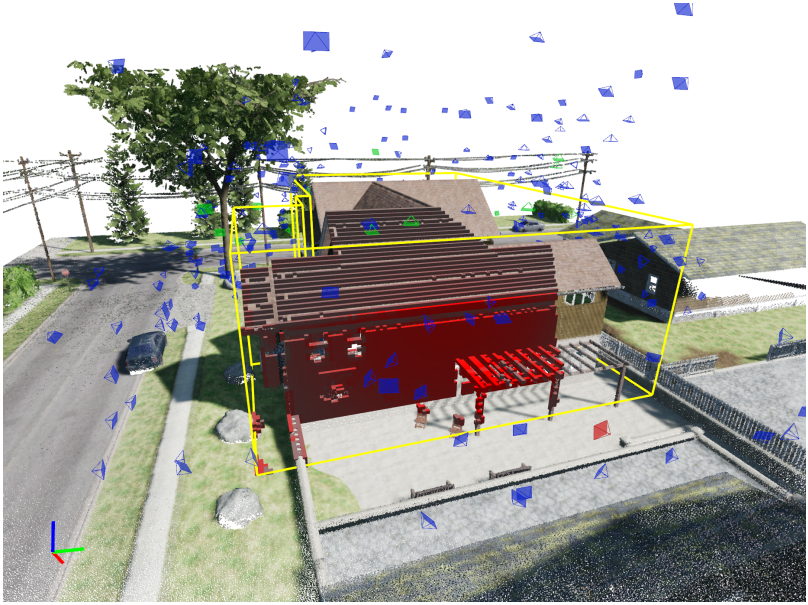

Figure 4.9: Visualization of the ray-cast obtained from a single camera viewpoint. The corresponding camera viewpoint is colored in red (lower right part of the image) and the occupancy map within the region of interest is rendered in gray. Voxels that are hit by the ray-cast are colored from gray to red with red indicating a higher value of $vi(\tau, v)$. Also shown is a dense point cloud of the scene and the region of interest as yellow lines.

dence factor $vi_i(\tau, v)$ to 0.2. This incentivizes multiple observations of the voxel for these ambiguous situations. An alternative to rendering the Poisson mesh from each viewpoint would be to compute a single view-independent normal for each voxel by searching for the nearest triangle in the Poisson mesh. However, this has certain limitations such as forcing a single normal for a voxel that is part of a crease of the surface. With our view-dependent normal computation the voxel can have different normals depending from which side it is viewed.

To accelerate the computation of the contributed information by a viewpoint we perform the ray-casting operation on a GPU using an image plane with a resolution of 600×450 and using a focal length of $f = 345$px, equivalent to that of the physical camera. We assume rays have an infinitesimal width and shoot a ray through the center of each pixel of the image plane. In our current setting a single voxel with edge-length of $0.2m$ projects onto one pixel at a distance of $\frac{0.2m \cdot f}{1\text{px}} \simeq 70m$.

Free-space motion paths

The final step in computing the viewpoint candidate graph G is to connect nodes via traversable edges (see Alg. 4 in Appendix)). Traversable refers to a path along which the drone's bounding box does not intersect with any non-free voxel. To this end we move the bounding box along the path with a maximum step-length of l, bounding the violation of the obstacle check to a maximum distance of $\sqrt{3}l$. By choosing the bounding box dimensions appropriately, we can ensure, up to the limits of the initial reconstruction, that the path is obstacle free and can be flown safely. For computational efficiency, we first attempt to connect viewpoints via straight lines. If an obstacle is encountered, we employ the rapidly exploring random trees (RRT*) algorithm [7] to find a free-space motion consisting of piecewise linear segments. A voxel is determined to be in free space if its occupancy is below the threshold oc_{free} (see Appendix for more details).

Sparse Matching

On most scenes, and inline with findings from the MVS literature, we attain high quality dense depth maps from relatively few images. However, in building-scale scenes clusters of views can be very far apart, making global registration of depth maps impossible. In order to account for this issue, we furthermore include a heuristic that will introduce additional viewpoints for registration and naturally integrates into our viewpoint candidate graph. To this end, we define a criterion based on the visible voxels in each viewpoint. To speed up the computation, we render the occupancy map at a lower resolution and count the voxels $obs(v)$ visible from a viewpoint v. Two viewpoints v_1 and v_2 are then called *matchable* if:

$$|obs(v_1) \cap obs(v_2)| \geq \alpha(|obs(v_1)| + |obs(v_2)|)/2 \quad , \tag{4.8}$$

ensuring that there is enough overlap of visible voxels between two viewpoints. We only insert motions into the viewpoint graph G if the two connected viewpoints are *matchable*. Incorporating this matching heuristic has proven to be very effective in our synthetic and real-world experiments. In all our experiments, we use a value of $\alpha = 0.4$. Once a matchable motion path is found we add an edge with a weight equal to the motion distance to the graph G.

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

Algorithm 2: Recursive greedy algorithm to maximize the optimization problem in Eq. Eq. (4.1). The recursive procedure takes the set of viewpoints V currently on the path VP (i.e. $V = \{v \; \forall \; v \in VP\}$) and the start viewpoint V_s, end viewpoint V_e and budget B of subproblem as input. It returns the viewpoint path for the subproblem. The initial call of the procedure is *recursiveGreedy*$(\emptyset, V_i, V_i, B_{total})$, where V_i is the viewpoint with the overall maximum score and B_{total} is the full travel budget of the quadrotor. The middle viewpoint V_m can be computed in an efficient manner by keeping a sorted list of IG and evaluating them in a lazy fashion (see Appendix for more details). A viewpoint is reachable with the current travel budget if the travel distance from V_s to V_m and from V_m to V_e does not exceed the budget B.

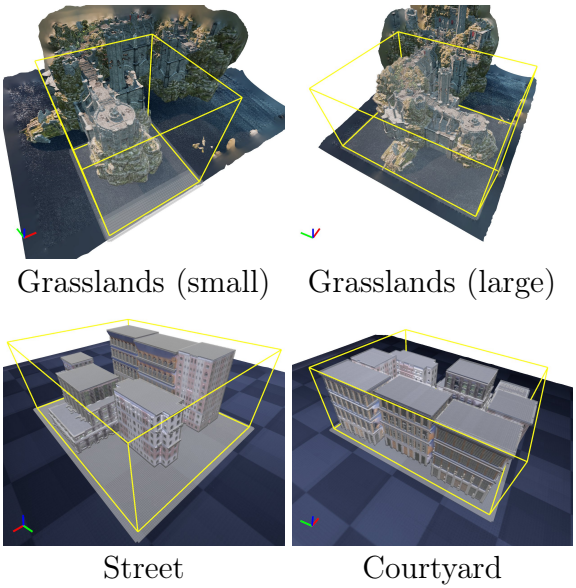

Figure 4.10: Overview of synthetic scenes used for evaluation. Shown in yellow is the bounding box of the region of interest together with the transparent voxel representation and the mesh representation of the scene. The *Grasslands* scenes on the left demonstrates interesting geometry that is mostly observable from easily accessible viewpoints. In contrast the *Street* and *Courtyard* scenes contain simpler geometry but many surfaces can only be observed from viewpoints between the buildings which are difficult to access.

4.5 Results

In this section, we discuss experiments conducted to evaluate our method both quantitatively and qualitatively, on real and synthetic data. Evaluating methods for building-scale robotic 3D reconstruction poses many significant challenges. First, comparative evaluation on *real* scenes is challenging due to changing conditions such as lighting, weather conditions, surrounding objects and modifications of the object itself that can occur over the course of hours. Furthermore, groundtruth information is typically not available for building size scenes. Therefore we report qualitative results from three challenging real-world examples, a renaissance *church* scene, an *office* building and a *historic* building. We also report quantitative results from a thorough evaluation on *synthetic* scenes from a state of the art rendering engine*.

The method proposed in here consists of many parts such as the robotic platform itself, the planning algorithm and the SfM/MVS pipeline. To unpack the influence of our core contribution (the planning algorithm) we compare our method with several strong baselines, including a state-of-the art method [80], in an ablative manner. First, we repeat the experiment in [80] to compare results from the end-to-end reconstruction of the full system *including* the SfM/MVS pipeline. Next we evaluate the relative performance to [80] with respect to the main optimization objective of achieved total viewpoint information (score). Finally, we compare our method with several strong baselines using reconstructions from recorded depth images to remove the influence of the SfM/MVS pipeline.

4.5.1 Synthetic scenes

We evaluate our method on four *synthetic* scenes stemming from the *Infinity Blade: Grass lands* environment† and the *Urban City* envi-

*http://www.unrealengine.com

†https://www.unrealengine.com/marketplace/infinity-blade-plain-lands

ronment[‡] from the Unreal Engine Marketplace. In the Grass lands environment we use a small region of interest as used in [80] and a large region of interest. The scenes from the Urban City environment are blocks of houses where the inner facades of the building are difficult to observe. An overview of the scenes and the regions of interest is shown in Fig. 4.10.

To evaluate our algorithm's performance when reconstructing a dense point cloud we closely follow the procedure established in [73]. We resample the ground truth mesh until the area of each triangle is below a certain threshold ($0.5m \times 0.5m$). From each triangle 100 points are sampled and the resulting point cloud is resampled on a voxel grid with a voxel size of $vs/2 = 0.05m$ by taking the mean of the points found within the same voxel. The resulting point cloud represents our ground truth. The reconstructed point cloud is resampled on the same voxel grid. To compare both point clouds we compute two quantities, the precision P and the recall R, where precision quantifies how many reconstructed points are close to a ground truth point and recall quantifies how many ground truth points are close to a reconstructed point. A point is close to another point if their distance is less or equal to $\delta = 0.1m$. Both quantities can be combined in the F1 score $F = 2\frac{P \cdot R}{P+R}$ which is a common performance measure for binary classification. We refer the reader to [73] for more details.

4.5.2 Comparison with Roberts et al.

Here we compare our reconstruction results with our reimplementation of [80] using our objective function. Like the authors we use Gurobi[§] to solve the resulting mixed-integer linear program and allow a runtime of 10 minutes. We observe that running Gurobi for much longer times does not improve the resulting score significantly (see Appendix 4.9.3). We show reconstruction results using ground-truth depth maps *and* end-to-end results when performing dense reconstruc-

[‡] `https://www.unrealengine.com/marketplace/urban-city`

[§] `http://www.gurobi.com/`

tion using Multi-View-Environment (MVE) [13].

We perform the comparison on the same *Grasslands (small)* scene as in [80] and allow a travel budget of $900m$. [13] We use 20 images arranged in a circular pattern to compute the initial reconstruction. [80] As shown in Table 4.1 our method improves upon [80] in both experimental settings. In Fig. 4.11 we qualitatively compare the reconstruction results using the end-to-end dense reconstruction, showing that our method can recover more surface details.

Using ground-truth depth maps

Method	Precision	Recall	F-Score
Roberts et al. [80]	97.22	62.53	76.11
Ours	96.56	67.16	**79.22**

Using full SfM/MVS pipeline

Method	Precision	Recall	F-Score
Roberts et al. [80]	81.83	64.91	72.39
Ours	80.29	72.17	**76.02**

Table 4.1: Quantitative comparison of the final reconstruction on the synthetic scene *Grass lands (small)* with a small region of interest and a travel budget of $900m$. Here we included the images from the initial scan when performing reconstruction. The best F-Score value is highlighted in bold. Note that Roberts et al. [80] refers to our reimplementation using our objective function.

4.5.3 Viewpoint score comparison

Here we directly compare our optimization method with the approach in [80] using our objective function. Recall that the optimization objective is a submodular viewpoint score describing how much surface of a scene has been covered and the solution space is restricted by

a budget constraint. We use the *Grasslands (small)* scene and the *Grasslands (large)* scene and measure the achieved viewpoint score for different travel budgets. We ran the method from [80] for $10min$ while our method took less than $10min$ in all cases.

The results in Fig. 4.12 show that our method consistently achieves a higher viewpoint score compared to [80]. In particular for larger scenes we can observe an increased difference in scores, suggesting that our method scales better to larger scenes. This is supported by later experiments on different scenes where Gurobi was unable to find a solution within a runtime of 60 minutes unless we reduced the number of considered viewpoints.

4.5.4 Comparison with regular baseline patterns

We compare reconstruction results against strong baselines, including regular patterns such as circles, meanders and hemispheres of different size and our reimplementation of [80] using our objective function. Note that the hemisphere pattern is computed based on our knowledge of free-space and our viewpoint graph and as such is more advanced than patterns used in current commercial tools. For the simpler *one-shot* baseline patterns (i.e. circle and meander) we choose the number of images to record for the 3D reconstruction in a best practice manner. In the Appendix 4.9.3 we show that including additional images does not necessarily improve and indeed often degrades the reconstruction performance.

We use three different scenes, *Grasslands (large)*, *Street* and *Courtyard*. For the *Grasslands (large)* scene we allow a travel budget of $1500m$ and use a circle pattern with 20 viewpoints for the initial reconstruction. For the *Street* and *Courtyard* scenes we allow a travel budget of $2700m$ and use 30 images arranged in a meander pattern so that our volumetric mapping can carve away more space between the buildings. Unfortunately, these scenes have repetitive textures which lead to many erroneous artifacts when we run the MVS pipeline end-to-end which make a comparison meaningless. To mitigate this issue we use rendered depth images along the computed viewpoint path

and fuse them into a dense point cloud by restricting the maximum distance, incidence angle and requiring at least 3 nearby depth measurements to reconstruct a point. We compute the precision, recall and F-Score on the resulting dense point cloud as described above.

In Fig. 4.13 we show qualitative comparison of the reconstructions. The measured quantities are shown in Table 4.2, Table 4.3 and Table 4.4. Our method yields higher F-scores than all other methods. The advantage of our method compared to a relatively simple hemisphere pattern becomes apparent in the more realistic *Street* and *Courtyard* scenes where a hemisphere pattern can not provide viewpoints that cover the inner facades of the buildings as they are shielded by the opposite buildings. Our method in contrast puts viewpoints within the free space between the buildings and also surrounding them to cover all surfaces. This is reflected in the higher recall and F-score compared to the hemisphere patterns.

Method	Precision	Recall	F-Score
Small circle ($35m$ radius)	77.29	13.26	22.64
Large circle ($70m$ radius)	72.55	3.92	7.44
Small meander ($70m \times 70m$)	44.68	20.00	27.64
Large meander ($140m \times 140m$)	43.60	20.57	27.95
Small hemisphere ($60m$ radius)	85.64	51.40	64.24
Large hemisphere ($75m$ radius)	81.49	50.41	62.29
NextBestView / Greedy	90.63	44.56	59.74
Adaptation of Roberts et al. [80]	87.87	50.03	63.76
Ours	90.57	57.70	**70.49**

Table 4.2: Quantitative comparison of the final reconstruction on the synthetic scene *Grass lands (large)* with a large region of interest and a travel budget of $1500m$. Here we included the images from the initial scan when performing reconstruction. The best F-Score value is highlighted in bold.

Method	Precision	Recall	F-Score
Small circle ($50m$ radius)	86.98	15.88	26.86
Large circle ($75m$ radius)	83.21	6.50	12.06
Small meander ($85m \times 100m$)	61.48	11.34	19.15
Large meander ($100m \times 115m$)	66.09	10.60	18.26
Small hemisphere ($60m$ radius)	91.34	63.95	75.23
Large hemisphere ($75m$ radius)	90.06	42.35	57.61
NextBestView / Greedy	94.23	68.46	79.30
Adaptation of Roberts et al. [80]	93.60	70.42	80.37
Ours	93.92	71.69	**81.31**

Table 4.3: Quantitative comparison of the final reconstruction on the synthetic scene *Courtyard* with a travel budget of 2700 m. Here we included the images from the initial scan when performing reconstruction. The best F-Score value is highlighted in bold.
Note that we had to limit the number of viewpoints used in our implementation of Roberts et al. [80] to 3000. Otherwise Gurobi was not able to find any solution within a runtime of 60 minutes.

4.5.5 Real scenes

In this section we discuss results acquired from real scenes and give a qualitative comparison with baselines from regular patterns as used in commercial tools. All reconstructed models shown here were computed with COLMAP [14, 18]. which showed subjectively better results than MVE [13].

Church

Fig. 4.14 shows results for the *church* scene, acquired with a total of 160 images. The initial flight pattern uses 20 viewpoints arranged in an ellipse. Based on the initial reconstruction a viewpoint path with

Method	Precision	Recall	F-Score
Small circle ($50m$ radius)	84.37	16.91	28.17
Large circle ($75m$ radius)	77.96	6.06	11.24
Small meander ($85m \times 100m$)	56.94	12.01	19.83
Large meander ($100m \times 115m$)	60.38	11.15	18.82
Small hemisphere ($60m$ radius)	90.55	74.51	81.75
Large hemisphere ($75m$ radius)	86.48	35.59	50.43
NextBestView / Greedy	90.99	78.18	84.10
Adaptation of Roberts et al. [80]	88.19	77.17	82.31
Ours	94.12	82.28	**87.80**

Table 4.4: Quantitative comparison of the final reconstruction on the synthetic scene *Street* with a travel budget of 2700 m. Here we included the images from the initial scan when performing reconstruction. The best F-Score value is highlighted in bold.
Note that we had to limit the number of viewpoints used in our implementation of Roberts et al. [80] to 4000. Otherwise Gurobi was not able to find any solution within a runtime of 60 minutes.

140 viewpoints and a maximum flight time of 10 minutes was planned (see Fig. 4.1) and flown.

We also compare our final result with two baselines, a small ellipse (Baseline 1) and a large ellipse (Baseline 2) in In Fig. 4.14. Both patterns contain the same total of 160 images as ours. It is evident that the baselines are not able to recover the same amount of detail as ours. Furthermore, geometric concavities such as the portal are much better resolved by our method. We show a further view of the church in Fig. 4.15 where fine geometric details such as pane separators of the windows are visible. Note that we chose a constant number of viewpoints to provide a fair comparison. While the baseline methods could also record many more images this does often degrade performance as

we demonstrate in the Appendix 4.9.3.

Further results

We show results from two further real-world experiments in Fig. 4.15, an *office* building with a very regular geometry and a *historic* building in a cluttered urban area showing the versatility of our method. For better comparison of details some areas are highlighted and enlarged. For the sake of brevity, we only show the best baseline for comparison.

Note the additional geometric details in our result compared to the baseline, in particular around windows, the stairs and sharp corners of the *historic* building. Even for the more regular and locally smooth geometry of the office building we see improved detail for the objects on the roof of the building and also surrounding the windows and along the roof line.

In both cases the total number of viewpoints was 80 and the maximum flight time was 10 minutes. We show the viewpoint paths in Fig. 4.16.

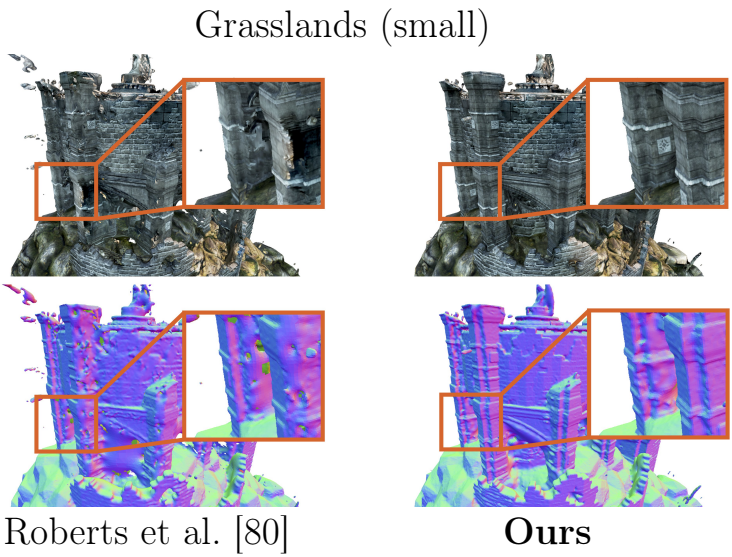

Figure 4.11: Qualitative comparison of our adaptation of [80] and our method on an end-to-end dense reconstruction of the Grasslands (smal l) scene. Here we included the images from the initial scan when performing reconstruction. The first row shows the geometry without color whereas the second row shows the texture mapped reconstruction. The method of [80] generates a viewpoint path that fails to capture some geometry information. Note that the missing geometry information also leads to distorted or erroneous textures. Note that Roberts et al. [80] refers to our reimplementation using our objective function.

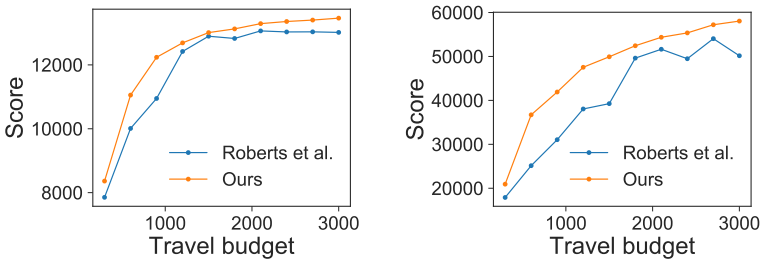

Figure 4.12: Comparison of submodular optimization methods on the synthetic scenes *Grasslands (small)* and *Grasslands (large)* using our objective function. Note that we would except both methods to saturate to the same value for an ever increasing budget. Thus we are in particular interested in the performance at intermediate budgets. As can be seen our method performs better for both scenes and all budgets but the improvement is much more pronounced in the larger scenes with more viewpoint candidates.

.

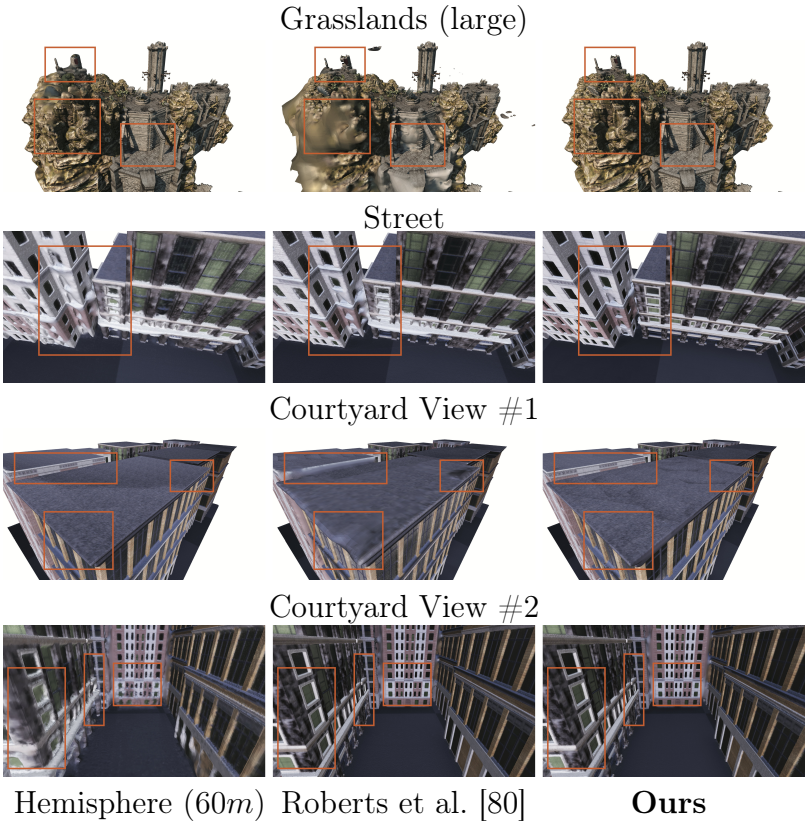

Figure 4.13: Qualitative comparison of reconstructed meshes. For different scenes we show results from a hemisphere pattern, our adaptation of [80] and our method. Here we included the images from the initial scan when performing reconstruction. Both the hemisphere pattern and [80] have missing points or lower point densities in some regions leading to bulging in the computed Poisson mesh. In contrast the reconstruction for our method shows less bulging. Compared to the hemisphere pattern our method shows more detailed texture and geometry in the lower facades of the building.

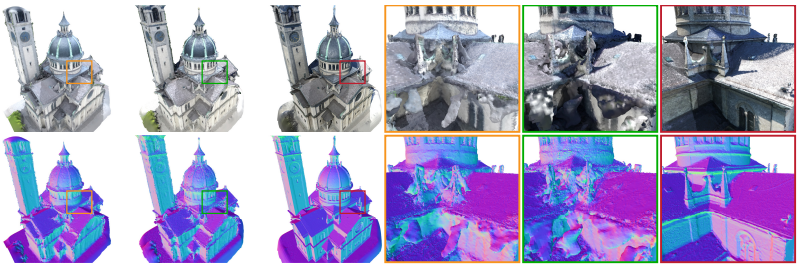

Figure 4.14: Our result for the *church* scene. The top row shows color renderings and the bottom row shows normal renderings. In particular in the normal renderings a lot of detailed structures are visible that are smoothed out or corrupted in baseline reconstructions.

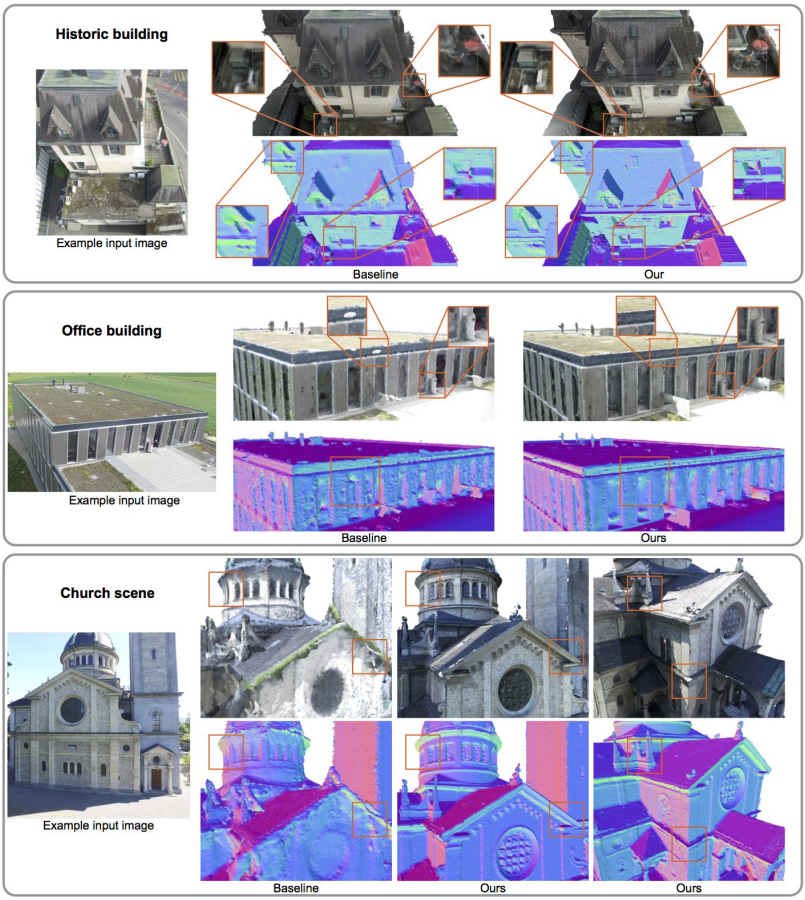

Figure 4.15: Results of our method. Top row: *Historic building* scene. Middle row: *Office building*. Bottom row: *Church* scene. Note that in all cases, the baseline and our approach use the same number of view points; 80, 80, and 160 views, respectively.

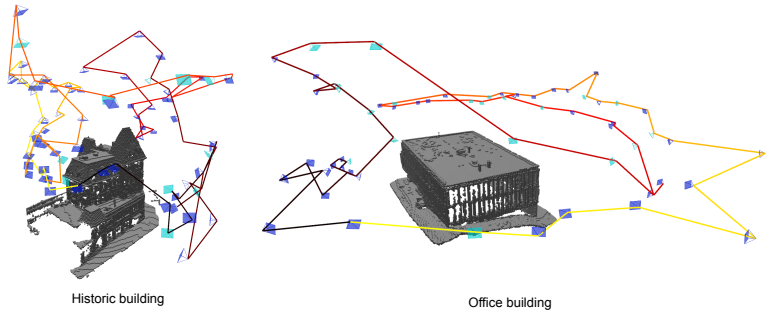

Figure 4.16: Our planned viewpoint trajectories for the *Historic building* and the *Office building*. The viewpoints selected by our algorithm are rendered in blue and the additional sparse matching viewpoints necessary to ensure a successful reconstruction are shown in cyan. The edges are color-coded to signal MAV progress along the path from black to red.

4.6 Discussion

We have proposed a method for the automated planning of viewpoints and safe free-space trajectories for acquisition of aerial images for dense monocular reconstruction. At the heart of our method lies a mathematical optimization formulation that explicitly reasons about observed and free, occupied and unobserved space. A volumetric occupancy grid is used to compute information gained from observing parts of the scene which in turn is used to compute an optimized set of viewpoints. Furthermore, the framework incorporates domain knowledge about SfM & MVS pipelines into the formulation such as preferring fronto-parallel views and ensuring viewpoints can be matched well. An approximate camera-model that allows us to decouple viewpoints from each other allows for a sub-modular and hence efficient implementation.

We have demonstrated the versatility of our method using a number of varied scenes ranging from a renaissance church to a modern office building. Our method produces high-quality reconstructions and can recover high-fidelity geometric and texture detail, and is capable of resolving even difficult parts of buildings such as concave portals, roofs and other overhangs. Furthermore, using synthetic scenes we have shown quantitatively that our method outperforms baselines such as a hemisphere pattern and the prior state-of-the-art.

As mentioned before our resulting reconstructions are not perfect. This is due to several issues that are at play: First, a SfM/MVS reconstruction is a complicated process that is very hard to model and thus our planned sequence of viewpoints does not necessarily lead to the expected reconstruction outcome based on our modeling assumptions. Second, the sequence of viewpoints is limited by the travel budget which was set to 900 seconds in this case. Third, the dynamic range in the images is limited and surfaces under the porch are very dark and might even saturate for different viewpoints. Fourth, The set of possible viewpoints is finite due to sampling and we also require some distances from obstacles to prevent collisions (in this case $1m$ distance from every mesh triangle in the scene, i.e. also $1m$ above the

ground). Finally, the dense point cloud from the MVS reconstruction might show inhomogeneous and low densities that lead to smooth arcs instead of sharp edges in the Poisson reconstruction of the mesh.

A major limitation of our work is the need to split the scanning procedure into two phases, i.e. we first coarsly scan the scene from a safe altitude, then perform necessary computations and planning and finally follow the planned trajectory to capture images for the final high quality reconstruction. The time required for computation and planning (in our case this is around $30 - 60 min$ dependent on the number of initial images and the desired quality level) often requires a second visit to the site which can be inconvenient or even prohibitive (if the scene or environmental conditions change quickly). While simple regular baseline patterns do not require a computation step the quality of the resulting reconstructions strongly depends on the structure of the scanned scene and for most buildings the need to fly at high altitudes (to avoid obstacles) results in steep viewing angles. We show a comparison of the required times in Sec. 4.9.4. Hence single-shot approaches with online planning or next-best-view selection would be desirable but may rely on robust and fast Structure from Motion and Multi View Stereo approaches both of which remain fruitful areas of future work.

4.7 Implementation details

Occupancy Map

As is typical in occupancy mapping [127, 9] we integrate depth maps into the occupancy map by updating the occupancy of each traversed voxel according to a beam-based inverse sensor model. We use an occupation probability for reflected beams of 0.7 and an occupation probability for transmitted beams of 0.4. Occupancy values are clamped to the range $[0.12, 0.97]$. A voxel is considered to be free if it's occupation probability is below $oc_{free} = 0.25$. We refer the reader to [9] for more details.

Recursive greedy method

In Alg. 2 the first step of the recursive procedure is to compute the middle viewpoint V_m for the next recursion level. This viewpoint must be reachable with the current budget, i.e. the sum of the travel distance from V_s to V_m and from V_m to V_e must not exceed the budget. Additionally, from these viewpoints we want to select the one with the maximum information gain given the current viewpoints on the path V.

To speed up this computation we keep a separate list that stores for each viewpoint a tuple of the viewpoint index and the corresponding information gain given the current viewpoint set. Initially the list contains the information gain given an empty viewpoint set and is sorted with descending information gains. Each recursion step receives a copy of this list so that subsequent updates only effect the following recursion steps. When searching for the next middle viewpoint we iterate through the list starting from the beginning. If a viewpoint is not reachable from the current V_s and V_e we skip it. Otherwise we update the information gain value and perform a single bubble sort iteration (i.e. push the entry back in the list) until the list is sorted again. We continue this until we reach a viewpoint that does not need to be reordered. This is the reachable viewpoint with maximum

information gain.

This procedure is well established for greedy submodular optimization [1]. As the information gain is submodular it can only decrease (or stay equal) with additional viewpoints. Thus, after recomputing the information gain of a viewpoint it can only move backwards in the list. Importantly, if we recompute the information gain and the viewpoint keeps a higher information gain than the next viewpoint it must be the viewpoint with the maximum information gain as all information gains of subsequent viewpoints can only decrease (or stay equal). This lazy evaluation strategy typically results in only a couple of information gain computations that are necessary. We observed speed ups in the order of $10 - 20$ compared to a non-lazy evaluation.

Dense Reconstruction Pipeline

We use standard structure from motion and multi-view stereo pipelines (MVE [13] for synthetic experiments and Colmap [14, 18] for outdoor experiments) to generate our models. We capture images by flying a DJI Matrice 100 drone with an onboard-computer. After landing, we perform structure from motion and multi-view stereo computations and filter the resulting depth-maps spatially by fusing the point-clouds, yielding a geo-referenced 3D model via incorporation of the image GPS coordinates. The depth maps are ray-casted into an occupancy map as described in Sec. 4.3. In cases where our model was initialized from a previous iterations, we fuse data into a single occupancy map. Finally, we generate a mesh from the dense point cloud using Screened Poisson Reconstruction [10].

Trajectory flight

Our planned trajectories are piecewise linear segments and the trajectories are flown via pure pursuit path tracking [128] with a desired velocity of $3m/s$ and a short lookahead distance of $2m$ leading to smooth motion. Note that we slow down at each viewpoint to take high resolution images without motion blur (i.e. when the next viewpoint is

within a $3m$ distance we reduce the desired velocity to $0.5m/s$). We consider this in our viewpoint path optimization by adding a heuristic budget cost of $3s$ for each viewpoint.

4.8 Algorithms for viewpoint graph generation

The formal description for our viewpoint generation and motion computation can be found in Alg. 3 and Alg. 4, respectively. The procedure *addViewpointAtPosition* samples an orientation and adds the resulting viewpoint to the viewpoint graph and the exploration front. An orientation is sampled by sampling a yaw and a pitch angle (roll is fixed to zero). Both angles are sampled in the same manner by first computing the angle pointing towards the center of the region of interest and then adding an offset. The offset is sampled from a zero-mean gaussian with a standard deviation equal to the angular range resulting from the axis-aligned bounding box of the region of interest.

```
Procedure generateViewpoints is
    Input: List of initial viewpoints V_0
    Input: Minmum number of viewpoints min_viewpoints
    Output: List of viewpoints V
    exploration_front ← V_0;
    while |exploration_front| > 0 or |V| < min_viewpoints do
        if |exploration_front| > 0 then
            ref_vp ← random sample from V;
            exploration_front ← V \ ref_vp;
            for direction ← 6 axial directions do
                step_size ← computeStepSize(ref_vp);
                p ← ref_vp.p + step_size * direction;
                addViewpointAtPosition(p);
            end
        else
            p ← random sample from allowed_space;
            addViewpointAtPosition(p);
        end
    end
end
```

Algorithm 3: Shown is the algorithm for generation the viewpoint candidate graph. The term *axial directions* refers to the $-x$, $+x$, $-y$, $+y$, $-z$, $+z$ directions.

```
Procedure findMotions is
    Input: List of viewpoints V
    Input: Number of neighbours to consider K
    Output: Dictionary M : {v1, v2} → m mapping two
            viewpoints to a motion m
    for v1 ∈ V do
        N ← find K nearest neighbours of v1;
        for v2 ∈ N do
            m ← findMotion(v1, v2);
            if m is a motion then
                M ← M ∪ {{v1, v2} : m};
            end
        end
    end
end
```

Algorithm 4: Shown is the algorithm for finding free-space motions between viewpoints in the viewpoint candidate graph.

4.9 Additional results

4.9.1 Submodular optimization results

To estimate the effect of runtime length of Gurobi for the baseline method [80] we ran Gurobi for selected travel budgets ($900m$, $1800m$) on the Grassland scene with small region of interest for 60 minutes. The resulting improvement in score was less than 1% in both cases.

4.9.2 Performance comparison when not including images from initial coarse scan

Here we report additional results when not including the images from the initial coarse scan for performing the 3D reconstruction. On the *Grasslands (small)* scene the F-score achieved by [80] is $F = 71.74$ ($P = 91.12$, $R = 59.15$) whereas our method improves upon this with an F-score of $\mathbf{F = 76.95}$ ($P = 89.64$, $R = 67.41$). In Fig. 4.18 we qualitatively compare the reconstruction results, showing that our method can recover more surface details.

In Tab. 4.5, Tab. 4.6 and Tab. 4.7 we show quantitative results for our experiments when not including the images from the initial coarse scan.

The corresponding qualitative results are shown in Fig. 4.19.

4.9.3 Effect of number of viewpoints for simple baseline methods

In our experiments we use a fixed number of viewpoints for the static baseline methods which do not require an initial scan (i.e. circle and meander patterns). To show that our choice of a limited number of viewpoints does not lead to an unfair bias we compare reconstructions resulting from different numbers of viewpoints in Tab. 4.8. The results show that for all patterns but the large meander pattern (which performs poorly in both cases) the performance is lower when using the larger number of viewpoints.

Method	Precision	Recall	F-Score
Small circle ($35m$ radius)	77.29	13.26	22.64
Large circle ($70m$ radius)	72.55	3.92	7.44
Small meander ($70m \times 70m$)	44.68	20.00	27.64
Large meander ($140m \times 140m$)	43.60	20.57	27.95
Small hemisphere ($60m$ radius)	86.43	46.84	60.76
Large hemisphere ($75m$ radius)	79.35	48.77	60.41
NextBestView	91.53	22.08	35.58
Adaptation of Roberts et al. [80]	88.61	46.44	60.94
Ours	89.72	50.70	**64.79**

Table 4.5: Quantitative comparison of the final reconstruction on the synthetic scene *Grass Lands* with a large region of interest and a travel budget of $1500m$. Here we did not include the images from the initial scan when performing reconstruction. The best F-Score value is highlighted in bold.

4.9.4 Comparison of times for different methods

In table Table 4.9 we show approximate required times for the simple one-shot methods and the other methods requiring an initial flight. Note that the one-shot methods take less field-experiment time than the other methods. However, the resulting reconstructions also have a lower quality and just flying longer does not allow us to increase the quality as shown in the previous section.

Method	Precision	Recall	F-Score
Small circle ($50m$ radius)	86.98	15.88	26.86
Large circle ($75m$ radius)	83.21	6.50	12.06
Small meander ($85m \times 100m$)	61.48	11.34	19.15
Large meander ($100m \times 115m$)	66.09	10.60	18.26
Small hemisphere ($60m$ radius)	91.07	66.35	76.77
Large hemisphere ($75m$ radius)	90.19	44.03	59.17
NextBestView	94.58	63.67	76.32
Adaptation of Roberts et al. [80]	92.93	72.15	81.23
Ours	93.59	74.30	**82.84**

Table 4.6: Quantitative comparison of the final reconstruction on the synthetic scene *Courtyard* with a travel budget of 2700 m. Here we did not include the images from the initial scan when performing reconstruction. The best F-Score value is highlighted in bold.
Note that we had to reduce the number of viewpoints used in the method from Roberts et al. [80] to 3000. Otherwise Gurobi was not able to find any solution within a runtime of 60 minutes.

Method	Precision	Recall	F-Score
Small circle ($50m$ radius)	84.37	16.91	28.17
Large circle ($75m$ radius)	77.96	6.06	11.24
Small meander ($85m \times 100m$)	56.94	12.01	19.83
Large meander ($100m \times 115m$)	60.38	11.15	18.82
Small hemisphere ($60m$ radius)	90.38	76.52	82.87
Large hemisphere ($75m$ radius)	86.65	37.25	52.10
NextBestView	94.97	75.42	84.07
Adaptation of Roberts et al. [80]	93.69	79.27	85.88
Ours	94.25	84.95	**89.36**

Table 4.7: Quantitative comparison of the final reconstruction on the synthetic scene *Street* with a travel budget of 2700 m. Here we did not include the images from the initial scan when performing reconstruction. The best F-Score value is highlighted in bold.
Note that we had to reduce the number of viewpoints used in the method from Roberts et al. [80] to 4000. Otherwise Gurobi was not able to find any solution within a runtime of 60 minutes.

Method	Precision	Recall	F-Score
Small circle ($25m$ radius, 120 views)	76.90	60.14	67.50
Small circle ($25m$ radius, 240 views)	52.72	48.69	50.63
Large circle ($50m$ radius, 120 views)	78.47	76.69	77.57
Large circle ($50m$ radius, 240 views)	50.27	61.48	55.31
Small meander ($50m \times 50m$, 144 views)	64.13	50.18	56.31
Small meander ($50m \times 50m$, 256 views)	51.11	39.10	44.31
Large meander ($75m \times 75m$, 144 views)	49.53	12.27	19.67
Large meander ($75m \times 75m$, 256 views)	47.20	14.01	21.60

Table 4.8: Quantitative comparison of the final reconstruction on the synthetic scene *Grass Lands* with a small region of interest. Shown are results for the one-shot baseline methods with different numbers of viewpoints. One can observe that the performance of the MVS reconstruction often decreases when more than 200 viewpoints are included.

Method	Flight time	Computation time	Total time
One-shot	$10min$	$0min$	$10min$
With initial scan	$20min$	$30 - 60min$	$50 - 80min$

Table 4.9: *Flight and experiment times for the different methods compared in this thesis.* Shown are estimated flight, computation and total field experiment times for the one-shot methods and the other methods requiring an initial scan. Note that the computation time depends on the specific scene. As we can see the one-shot method takes much less time.

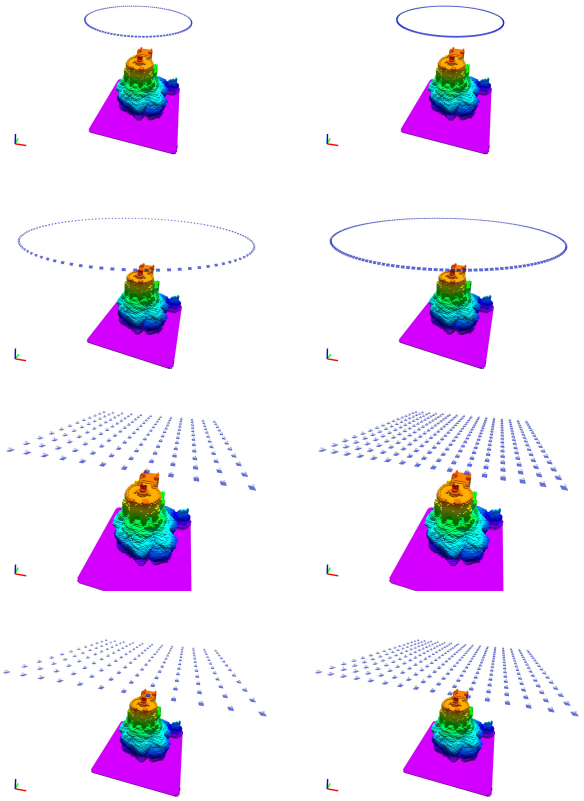

Figure 4.17: Viewpoints used for comparison in Table 4.8. Shown from top to bottom: Circle with 25m radius, circle with 50m radius, Meander with 50m side length, meander with 75m side length. From left to right: Medium number of viewpoints, high number of viewpoints.

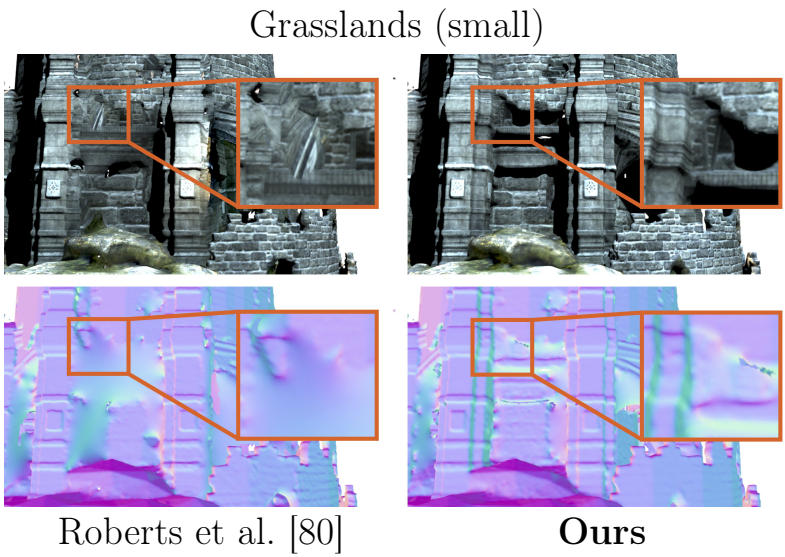

Figure 4.18: Qualitative comparison of our adaptation of [80] and our method on an end-to-end dense reconstruction of the Grasslands (smal.) scene. Here we did not include the images from the initial scan when performing reconstruction. The first row shows the geometry without color whereas the second row shows the texture mapped reconstruction. The method of [80] generates a viewpoint path that fails to capture some geometry information. Note that the missing geometry information also leads to distorted or erroneous textures. Note that Roberts et al. [80] refers to our reimplementation using our objective function.

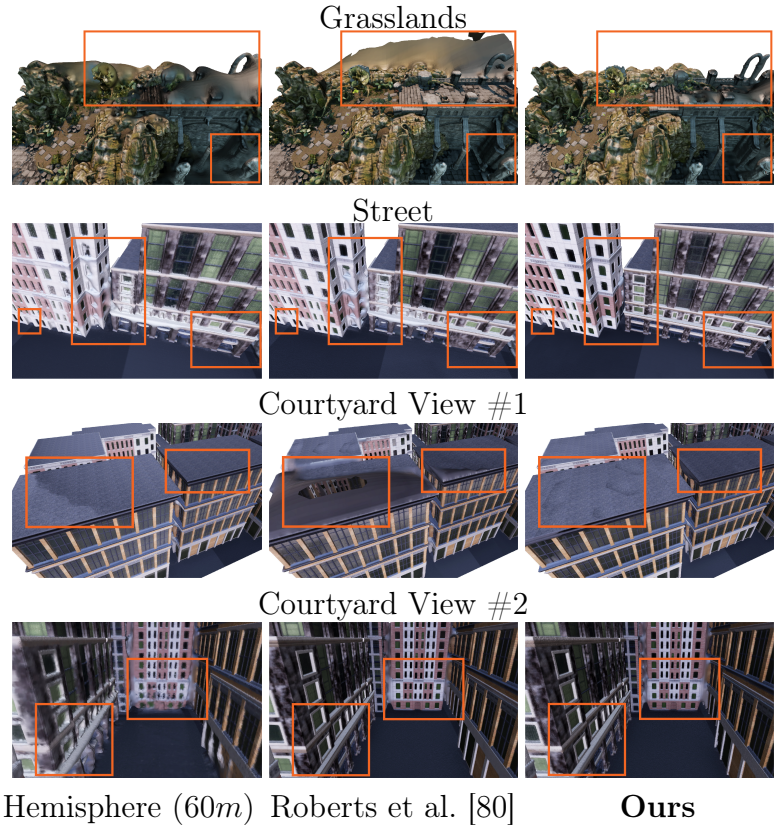

Figure 4.19: Qualitative comparison of reconstructed meshes. For different scenes we show results from a hemisphere pattern, our adaptation of [80] and our method. Here we did not include the images from the initial scan when performing reconstruction. Both the hemisphere pattern and [80] have missing points or lower point densities in some regions leading to bulging in the computed Poisson mesh. In contrast the reconstruction for our method shows less bulging. Compared to the hemisphere pattern our method shows more detailed texture and geometry in the lower facades of the building.

Chapter 5

Learning a Viewpoint Utility Score

The system in chapter 4 can compute flight plans that result in compelling 3D reconstructions it has the limitation of requiring an initial flight to generate a coarse prior 3D model. This can be undesirable in certain situations such as limited or difficult access to the area of interest or strong time constraints. Here we look at the problem of reconstructing and exploring an unknown scene without requiring an initial scan. This chapter is based upon our work in [129] .

When free space in the scene is approximately known, an offline planner can generate optimal plans to efficiently explore the scene. However, for exploring unknown scenes, the planner must predict and maximize usefulness of where to go on the fly. Traditionally, this has been achieved using handcrafted utility functions. We propose to learn a better utility function that predicts the usefulness of future viewpoints. Our learned utility function is based on a 3D convolutional neural network. This network takes as input a novel volumetric scene representation that implicitly captures previously visited viewpoints and generalizes to new scenes. We evaluate our method on several

large 3D models of urban scenes using simulated depth cameras. We show that our method outperforms existing utility measures in terms of reconstruction performance and is robust to sensor noise.

5.1 Introduction

Quadrotors, drones, and other robotic cameras are becoming increasingly powerful, inexpensive and are being used for a range of tasks in computer vision and robotics applications such as autonomous navigation, mapping, 3D reconstruction, reconnaissance, and grasping and manipulation. For these applications, modeling the surrounding space and determining which areas are occupied is of key importance.

Recently, several approaches for robotic scanning of indoor [130] and outdoor [131, 68] scenes have been proposed. Such approaches need to reason about whether voxels are free, occupied, or unknown space to ensure safety of the robot and to achieve good coverage w.r.t. their objective function (e.g. coverage of the 3D surfaces [131]). Model-based approaches require approximate information about free space and occupied space, which is typically acquired or input manually. This prevents such approaches from being fully autonomous or deployed in entirely unknown scenes [132]. Model-free approaches can be applied in unknown environments [133, 110, 134, 135]. Irrespective of the type of approach used, all algorithms require a utility function that predicts how useful a new measurement (i.e. depth image) would be. Based on this utility function a planner reasons about the sequence of viewpoints to include in the motion plan. This utility function is often a hand-crafted heuristic and hence it is difficult to incorporate prior information about the expected distributions of 3D geometry in certain scenes.

We propose to devise a better utility function using a data-driven approach. The desired target values for our utility function stem from an oracle with access to ground truth data. Our learned utility function implicitly captures knowledge about building and geometry distributions from approporiate training data and is capable of pre-

dicting the utility of new viewpoints given only the current occupancy map. To this end we train a 3D ConvNet on a novel multi-scale voxel representation of an underlying occupancy map, which encodes the current model of the environment. We then demonstrate that the learned utility function can be utilized to efficiently explore unknown environments.

The input to our network relies only on occupancy and hence abstracts away the capture method (i.e. stereo, IR depth camera, etc.). While our training data consists of perfect simulated depth images we demonstrate in our experiments that our learned model can be used with imperfect sensor data at test time, such as simulated noisy depth cameras or stereo data. The approach is not limited to environments with a fixed extent and generalizes to new scenes that are substantially different from ones in the training data. Our approach outperforms existing methods, that use heuristic-based utility functions [132, 134, 135] and is more than 10× faster to compute than the methods from [134, 135].

5.2 Related work

Exploration and mapping are well studied problems. We first discuss theoretical results and then describe approaches in the active vision domain and finally work in 3D vision.

Submodular sensor placement: In the case of a priori known environments and a given set of measurement locations, much work is dedicated to submodular objective functions for coverage [136, 25]. Submodularity is a mathematical property enabling approximation guarantees on the solution using greedy methods. While work exists on dynamic environments where the utility of future measurements can change upon performing a measurement [137, 138], these methods are usually difficult to scale to large state and observation spaces, which we considered in this chapter as they are common in computer vision applications.

Next-best-view and exploration: In the next-best-view setting, the set of measurement locations is often fixed a priori as in the submodular coverage work described above. The work in this area is usually concerned with defining good heuristic utility functions and approximating the coverage task to make it computationally feasible [78, 110, 111, 112, 113]. A number of heuristics is explicitly compared in [134, 135], and a subset of these is computed and used as a feature vector by Choudhury *et al.* [139] to imitate an approximately optimal strategy with ground-truth access.

Based on an a priori fixed set of camera poses and a binary input mask of already visited poses Devrim *et al.* [140] use reinforcement learning to regress a scalar parameter used in the selection algorithm for the next view. In contrast to our work the approach is concerned with a priori known, fixed environments and camera poses making it suitable for inspection planning.

In active vision, a large body of work is concerned with exploration through only partially known scenes. Frontier-based algorithms [106] are used for autonomous mapping and exploration of the environment using stereo [107], RGB-D, or monocular cameras [108]. Heng *et al.* [133] propose a method which alternates between exploration and optimizing coverage for 3D reconstruction.

All of the approaches discussed above either define or are based on heuristics to decide on the utility of the next measurement or require prior knowledge of environment and possible camera poses. Instead of hand-crafting a utility function our work is concerned with learning such a function that can outperform existing hand-crafted functions and is computationally cheaper to evaluate. Additionally, our approach does not need a priori knowledge of the map.

3D convolutional neural networks: A large body of work in computer vision is concerned with processing of 3D input and output data using convolutional neural networks. In some cases this data stems from RGB-D images such as in Song *et al.* [141] where the goal is to detect objects. In other contexts, volumetric input in the form of binary occupancy labels or signed distance functions are used

for diverse tasks such as shape classification and semantic voxel labeling [142, 143], surface completion [144], hand pose estimation [145], or feature learning [146]. These works are concerned with passive tasks on uniform input grids of fixed dimensions, containing the object or surface of interest. This prevents reasoning across large distances or requires one to reduce the level of detail of the input.

Different representations of occupancy grids have been proposed to mitigate the trade-off of large uniform input dimensions and level of detail [143]. However, in the context of our work the occupancy map is often not very sparse as it is generated by casting rays into a tri-state map and updating continuous values which results in very few homogeneous regions which would benefit from the formulation by Riegler *et al.* [143]. Also related to our work are approaches to multi-view reconstruction [147] where the output is predicted based on a sequence of input images. In contrast to our work Liu *et al.* [148] reconstruct small objects in a fixed size volume whereas we are concerned with large city scenes containing several buildings.

5.3 Problem Setting and Overview

Our work is concerned with the automatic exploration of an a priori unknown 3D world with the ultimate goal of reconstructing the surfaces of the scene in an efficient manner. To this end the exploring agent executes a loop of alternating sense and act operations as depicted in Fig. 5.1. After a measurement is taken (i.e. in the form of a depth image) it is integrated into an occupancy map. A planner (described in Section 5.4.4) has a set of reachable candidate viewpoints and queries a utility function for each of them. The utility function in turn provides a score for each viewpoint indicating how useful the viewpoint is for the exploration task. The planner then decides to move to the viewpoint with the highest score. Here we focus on the utility function and its corresponding input representation.

In Fig. 5.2 we illustrate the desired characteristic of the utility function. Note that the utility function has to score a viewpoint only based

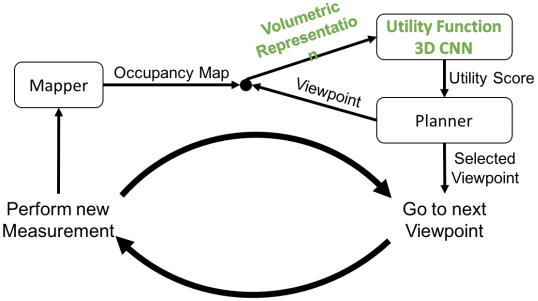

Figure 5.1: This diagram shows the exploration loop. After performing a new measurement (i.e. a depth image) the map is updated. This map is used to generate a volumetric representation that is fed into the utility function (in our case a learned 3D CNN). The planner can query the utility function on a set of candidate viewpoints. Based on the computed utility scores the planner then select the viewpoint to which the agent will move and perform a new measurement. In this chapter we focus on the parts depicted in green: The volumetric representation and the corresponding learned utility function.

on the current map information. In Fig. 5.2 the camera is surrounded by some space, already known to be free (white) and part of the surface has been observed (blue). The next viewpoint is restricted to the known free space, whereas moving into unknown space (light green) could lead to collisions. The main difficulty stems from the fact that the utility function needs to predict how much unknown surface can be discovered from a new viewpoint. Much work has been dedicated to developing and studying various heuristics to compute a score that quantifies the expected value of possible viewpoints [134, 135].

We propose a data-driven approach where we use supervised learn-

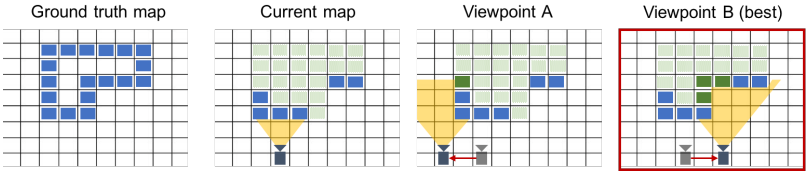

Figure 5.2: The exploration task (here depicted in 2D for clarity) is to discover occupied surface voxels (shown here in blue). Voxels are initially unknown (shown here in light green) and get discovered by taking a measurement, e.g., shooting rays from the camera into the scene. Voxels that are not surface voxels will be discovered as free voxels (shown here in white). Each possible viewpoint has a corresponding utility value depending on how much it contributes to our knowledge of the surface (shown here in dark green). To decide which viewpoint we should go to next, an ideal utility score function would tell us the expected utility of viewpoints before performing them. This function can then be used in a planing algorithm to visit a sequence of viewpoints with the highest expected utility.

ing to find a utility function that imitates an oracle. The oracle has access to the ground truth map and can compute the true utility score. For this task we introduce a map representation consisting of multi-scale sub-volumes extracted around the camera's position. For all possible viewpoints this data is fed into a 3D ConvNet at training time together with the oracle's score as a target value. Intuitively, the model learns to predict the likelihood of seeing additional surface voxels for any given pose, given the current occupancy map. However, we do not explicitly model this likelihood but instead provide only the oracle's score to the learner. We experimentally show that our formulation generalizes well to new scenes with different object shape

and distribution and can handle input resulting from noisy sensor measurements.

We follow related work [134, 135, 140] and evaluate our method on simulated but high-fidelity environments. This allows for evaluation of the utility function and reduces the influence of environmental factors and specific robotic platforms. Our environments contain realistic models of urban areas in terms of size and distribution of buildings. Furthermore it is important to note that our technique only takes occupancy information as input and does not directly interface with raw sensor data. In addition we test our approach on real data from outdoor and indoor scenes to demonstrate that our method is not limited to synthetic environments.

5.4 Predicting View Utility

We first formally define our task and the desired utility function and then introduce our method for learning and evaluating this function.

5.4.1 World model

We model the world as a uniform voxel grid V with resolution r. A map M is a tuple $M = (M^o, M^u)$ of functions $M^o : V \rightarrow [0, 1]$, $M^u : V \rightarrow [0, 1]$ that map each voxel $v \in V$ to an occupancy value $M^o(v)$ describing the fraction of the voxel's volume that is occupied and an associated uncertainty value $M^u(v)$, i.e. 1 for total uncertainty and 0 for no uncertainty. Maps change over time so we denote the map at time t as M_t.

After moving to a viewpoint $\mathbf{p}$ the camera acquires a new measurement in the form of a depth image and the map M is updated. We denote the updated map as $M|_{\mathbf{p}}$. The uncertainty is updated according to

$$M^u|_{\mathbf{p}}(v) = \exp(-\eta) M^u(v) \quad , \tag{5.1}$$

where $\eta \in \mathbb{R}_{>0}$ describes the amount of information added by a single measurement. This is a simple but effective measure providing a diminishing information gain of repeated measurements. Note that $M^u|_{\mathbf{p}}(v) \leq M^u(v)$ so uncertainty decreases with additional measurements. As is typical in occupancy mapping [127, 9] we update the voxel occupancies $M^o(v)$ according to a beam-based inverse sensor model. Please see Sec. 5.4.4 for details on initialization of the map.

5.4.2 Oracle utility function

To select viewpoints, we need a utility function that assigns scores to all possible viewpoints at any time. We first introduce an oracle utility function with access to the ground truth (set of true surface voxels) during evaluation. It returns the desired true utility measure. We will then learn to imitate the oracle without access to ground truth.

We characterize a good viewpoint as one that discovers a large amount of surface voxels. Let *ObsSurf*(M) be the total number of observed surface voxels in map M weighted by their associated certainty value:

$$\mathit{ObsSurf}(M) = \sum_{v \in \mathit{Surf}} (1 - M^u(v)) \quad , \tag{5.2}$$

where $\mathit{Surf} \subseteq V$ is the set of ground truth surface voxels, i.e. all voxels that intersect the surface. Note that *ObsSurf*(M) increases monotonically with additional measurements because the certainty of voxels can only increase according to Eq. Eq. (5.1).

The decrease in uncertainty of surface voxels with a new measurement defines the oracle's utility score. We express this score as a function of the current map M and the camera pose $\mathbf{p}$:

$$\begin{aligned} s(M, \mathbf{p}) &= \mathit{ObsSurf}(M|_{\mathbf{p}}) - \mathit{ObsSurf}(M) \\ &= \sum_{v \in \mathit{Surf}} (-M^u|_{\mathbf{p}}(v) + M^u(v)) = \sum_{v \in \mathit{Surf}} (1 - \exp(-\eta))\, M^u(v) \geq 0 \quad . \end{aligned} \tag{5.3}$$

5.4.3 Learning the utility function

Computing the utility score introduced in Eq. 5.3 for any viewpoint requires access to the ground truth map. Our goal is to predict $s(M, \mathbf{p})$ without access to this data so we can formulate a regression problem that computes score values given occupancy maps as input.

Multi-scale map representation

We propose to make predictions directly based on the occupancy map, rather than based on a temporal sequence of raw inputs. This occupancy map encodes our knowledge of already observed surfaces and free space and ultimately can be used to build up a map for both navigation and 3D reconstruction.

For use in a 3D ConvNet the map has to be represented with fixed dimensionalities. Here a trade-off between memory consumption, computational cost, reach and resolution arises. For example, extracting a small high resolution grid around the camera would constrain information to a small spatial extent whereas a grid with large spatial extent would either lead to rapid increase in memory consumption and computational cost or would lead to drastic reduction in resolution.

To mitigate this issue we introduce a multi-scale representation by sampling the occupancy map at multiple scales as depicted in Fig. 5.3. For each scale $l \in \{1, \ldots, L\}$ we extract values on a 3D grid of size $D_x \times D_y \times D_z$ and resolution $2^l r$ (orange points in Fig. 5.3). On scale l the map values are given by averaging the 2^l closest voxels (gray rectangles in Fig. 5.3). This can be done efficiently by representing the map as an octree. The 3D grids are translated and rotated according to the measurement pose $\mathbf{p}$ and we use tri-linear interpolation of the map values to compute the values on the grid. This representation allows us to capture both coarse parts of the map that are far away from the camera but still keep finer detail in its direct surroundings. Furthermore, it provides an efficient data representation of fixed size, suitable for training of a 3D ConvNet. We denote the multi-scale

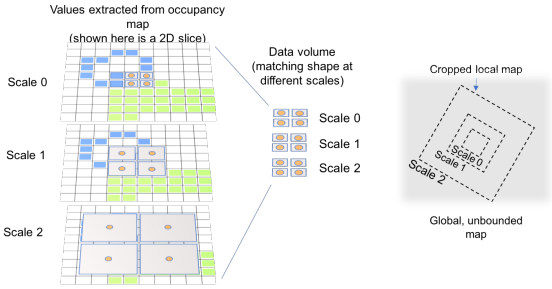

Figure 5.3: Local multi-scale representation of an occupancy map. For clarity of presentation we shows the 2D case for a grid of size 2×2. The occupancy map is sampled with 3D grids at multiple scales centered around the camera position. Sample points on different scales are shown in orange and their extent in gray.

representation by $x(M, \mathbf{p}) \in \mathbb{R}^{D_x \times D_y \times D_z \times 2L}$. Note that the factor 2 stems from extracting the occupancy and the uncertainty value on each scale.

In Fig. 5.5 we show visualizations of the multi-scale representation for a grid size of 16. Shown are horizontal slices of the 3D grid representation at a fixed height for random samples from the *Neighborhood* dataset. The multiscale representation allows us to get an idea about the surrounding of the camera pose: From Fig. 5.5 we can infer that there is a building at the lower right corner and on the larger scales we can even see that there is also a building in the upper left corner. We show visualization of other grid sizes in Fig. 5.4, Fig. 5.6 and Fig. 5.7. We selected a grid size of 16 as the best trade-off between captured detail and computation and storage requirements.

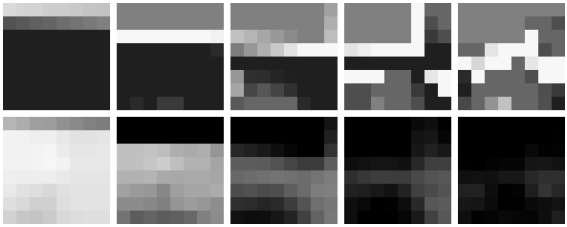

Figure 5.4: Example of multi-scale representation for grid size of 8. Shown are horizontal slices of the 3D grid representation at a fixed height. This example is a random sample from the *Neighborhood* dataset (grid size 8 with $L = 5$ scales). Top row: Occupancies. Bottom row: Certainties (i.e. $1 - v^u$). Left column: Smallest scale. Right column: Largest scale. Black represents 0, white represents 1.

ConvNet Architecture

We now describe our proposed model architecture used to learn the desired utility function $f : \mathbb{R}^{D_x \times D_y \times D_z \times 2L} \rightarrow \mathbb{R}$. The general architecture is shown in Fig. 5.8 and consists of a number N_c of convolutional blocks followed by two fully connected layers with ReLu activations. Each convolutional block contains a series of N_u units where a unit is made up of a 3D convolution, followed by Batch-Norm, followed by ReLu activation. Each 3D convolution increases the number of feature maps by N_f. After each block the spatial dimensions are downscaled by a factor of 2 using 3D max-pooling. The first fully connected layer has N_{h1} hidden units and the second one has N_{h2} hidden units. Note that we do not separate the input data at different scales so that the network can learn to combine data on different scales. More details on the exact architecture are provided in Sec. 5.5.1.

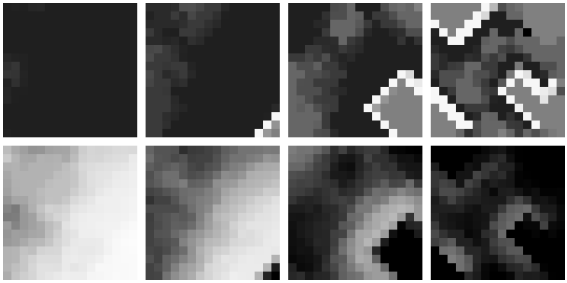

Figure 5.5: Example of multi-scale representation for grid size of 16. Shown are horizontal slices of the 3D grid representation at a fixed height. This example is a random sample from the *Neighborhood* dataset (grid size 16 with $L = 4$ scales). Top row: Occupancies. Bottom row: Certainties (i.e. $1 - v^u$). Left column: Smallest scale. Right column: Largest scale. Black represents 0, white represents 1.

We use a weight-regularized $L2$ loss

$$\mathcal{L}(X, Y; \theta) = \sum_{i=1}^{N} \|f(X_i) - Y_i\|_2^2 + \lambda \|\theta\|_2^2 \quad , \tag{5.4}$$

where θ are the model parameters, λ is the regularization factor and (X_i, Y_i) for $i \in \{1, \ldots, N\}$ are the samples of input and target from our dataset.

5.4.4 3D Scene Exploration

To evaluate the effectiveness of our utility function, we implement a next-best-view (NBV) planning approach, to sequentially explore a 3D scene. Here we provide details of our world model and our methods for execution of episodes for the data generation phase and at test time.

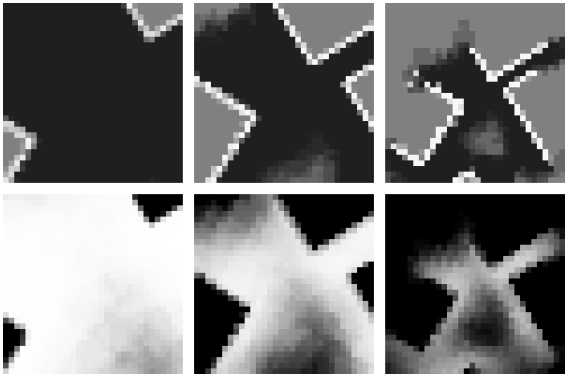

Figure 5.6: Example of multi-scale representation for grid size of 32. Shown are horizontal slices of the 3D grid representation at a fixed height. This example is a random sample from the *Neighborhood* dataset (grid size 32 with $L = 3$ scales). Top row: Occupancies. Bottom row: Certainties (i.e. $1 - v^u$). Left column: Smallest scale. Right column: Largest scale. Black represents 0, white represents 1.

We assume exploration of the world occurs in episodes. To initialize a new episode, the camera pose at time t_0 is chosen randomly in free space such that no collision occurs and the camera can move to each neighboring viewpoint without collision. A collision occurs if a bounding box of size $(1m, 1m, 1m)$ centered at the camera pose intersects with any occupied or unknown voxel. Initially, all voxels $v \in V$ are initialized to be unknown, i.e. $M^u_{t_0}(v) = 1, M^o_{t_0}(v) = v^{o,prior}\ \forall v \in V$, where $v^{o,prior}$ is a prior assumption on the occupancy and we use $v^{o,prior} = 0.5$ throughout this work. To enable initial movement of the camera we clear (i.e. set to free space) a bounding box of $(6m)^3$ around the initial camera position.

At each time step t, we evaluate each potential viewpoint with our utility function, and move to the viewpoint $\mathbf{p}^*(t)$ that gives the best

expected reward according to:

$$\mathbf{p}^*(t) = argmax_{\mathbf{p}\in P(t)} u(M_t, \mathbf{p}) \quad , \tag{5.5}$$

where $P(t)$ is the set of potential viewpoints and $u(\cdot)$ is the utility function in use.

At the start of each episode the set of potential viewpoints only contains the initial viewpoint. At each time step the set $P(t)$ is extended by those neighbors of the current viewpoint that do not lead to a collision. We ignore potential viewpoints if they have already been observed twice. Each viewpoint has 9 neighbors, 6 of them being positive and negative translations of $2.5m$ along each axis of the camera frame, 2 rotations of 25°, clock-wise and counter-clockwise along the yaw axis, and a full turnaround of 180° along the yaw axis. We keep pitch and roll angles fixed throughout.

After moving to a new viewpoint, the camera takes a measurement in the form of a depth image and the map is updated . Note that we use ground truth depth when generating training data but later demonstrate that we can use noisy depth images and even stereo depth at test time.

Note that we assume that the utility function is submodular. While this is true for the oracle utility it is not necessarily the case for other utility functions (i.e. our learned model). Nevertheless, this assumption allows us to perform lazy evaluations of the utility function [1] .

5.4.5 Dataset

To learn the utility function $f(x)$, approximating the oracle (see Eq. 5.3) we require labeled training data. Our data should capture large urban environments with a variety of structures typical for human-made environments. To this end we chose models from the *3D Street View* dataset [149]. These models feature realistic building distributions and geometries from different cities. Additionally, we chose a large scene from a photo-realistic game engine (`https://www.unrealengine.`

com) containing small buildings in a suburban environment, including trees, smaller vegetation and power lines. All environments are shown in Fig. 5.9. Note that we only use data from *Washington2* to train our model. While *Washington1* and *Paris* are similar in terms of building height the building distribution and geometry is clearly different. A particular challenge is posed by the *SanFrancisco* scene which includes tall buildings never seen before in *Washington2*. Similarly, the buildings and vegetation in the *Neighborhood* scene are unlike anything seen in the training data.

We generate samples by running episodes with $r = 0.4m$ until time $t_e = 200$ and selecting the best viewpoint $\mathbf{p}$ according to the oracle's score at each time step. For each step t we store tuples of input $x(M, \mathbf{p})$ and target value from the oracle $s(M, \mathbf{p})$ for each neighboring viewpoint.

Note that we record samples for each possible neighbor of the current viewpoint (instead of only the best selected viewpoint). This is necessary as our predictor will have to provide approximate scores for arbitrary viewpoints at test time. We record a total of approximately $1,000,000$ samples and perform a 80/20 split into training and validation set. To encourage future comparison we will release our code for generating the data and evaluation.

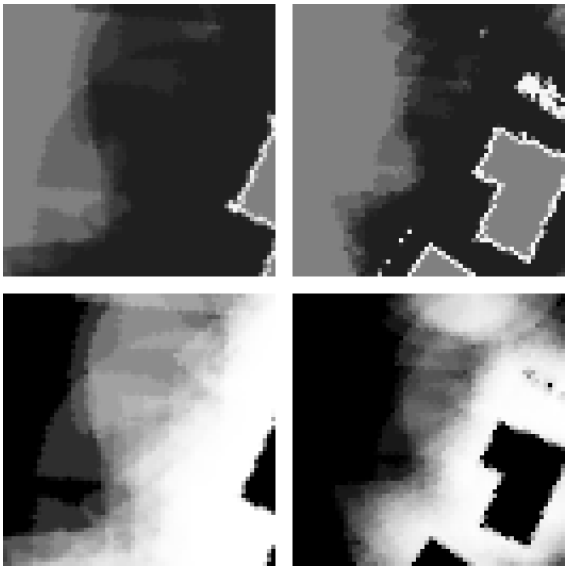

Figure 5.7: Example of multi-scale representation for grid size of 64. Shown are horizontal slices of the 3D grid representation at a fixed height. This example is a random sample from the *Neighborhood* dataset (grid size 64 with $L = 2$ scales). Top row: Occupancies. Bottom row: Certainties (i.e. $1 - v^u$). Left column: Smallest scale. Right column: Largest scale. Black represents 0, white represents 1.

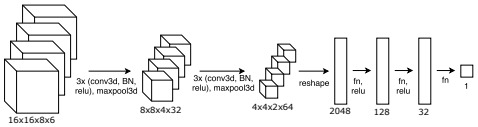

Figure 5.8: Our architecture for an input size of $16 \times 16 \times 8$ with $L = 3$ scales resulting in $2L = 6$ channels. The model consists of blocks (made up of multiple units each performing 3D convolution, batch-norm and ReLu) followed by downscaling using 3D max-pooling. This pattern is performed until we arrive at a data volume with spatial dimension $4 \times 4 \times 2$. This is reshaped into a single vector followed by two fully connected layers with ReLu activation and a final linear layer predicting a scalar score value.

Figure 5.9: Normal rendering of environments. From left to right: *Washington2*, *Washington1*, *Paris*, *SanFrancisco*, *Neighborhood*.

5.5 Experiments

We describe our ConvNet architecture and then show different evaluations of our method.

5.5.1 ConvNet architectures and training

We evaluated different ConvNet variants by varying N_c, N_u and N_f. We also tried modifications such as using residual units [150, 151]. Here we report results on the best performing model with input size $16 \times 16 \times 8$ ($N_c = 2$, $N_u = 4$, $N_f = 8$, $L = 3$, $N_{h1} = 128$, $N_{h2} = 32$), denoted as **Ours** in the rest of the section. Training of the model is done with ADAM using a mini-batch size of 128, regularization $\lambda = 10^{-4}$, $\alpha = 10^{-4}$ and the values suggested by Kingma *et al.* [152] for the other parameters. Dropout with a rate of 0.5 is used after fully-connected layers during training. Network parameters are initialized according to Glorot *et al.* [153] (corrected for ReLu activations). We use early stopping when over-fitting on test data is observed.

5.5.2 Evaluation

Our evaluation consists of three parts. First we evaluate our model on datasets generated as described in Sec. 5.4.5 and report spearman's rho to show the rank correlation of predicted scores and ground truth scores. Following this, we compare our models with previously proposed utility functions from [132, 134, 135]. We use the open-source implementation provided by [134, 135] and report results on their best performing methods on our scenes, *ProximityCount* and *AverageEntropy*. We also compare with a frontier-based function measuring the number of unobserved voxels visible from a viewpoint as in [133, 154]. For this comparison we use simulated noise-free depth images for all methods. Finally, we evaluate our models with depth images perturbed by noise and depth images produced by stereo matching in a photo-realistic rendering engine.

To demonstrate the generalization capability of our models we use four test scenes (column 2-5 in Fig. 5.9) that show different building distribution and geometry than the scene used to collect training data. We also perform the experiments on the training scenes where the exploration remains difficult due to random start poses and possible ambiguity in the incomplete occupancy maps.

To compute score and efficiency values, we run 50 episodes with $r = 0.4m$ until $t_e = 200$ for each method and compute the sample mean and standard deviation at each time step. To enable a fair comparison, we select a random start position for each episode in advance and use the same start positions for each method.

In order to report a single metric of performance for comparison we compute the area under the curve of observed surface versus time (see plots in Fig. 5.10):

$$\mathit{eff} = \sum_{t=0}^{t_e} \mathit{ObsSurf}(M_t) \quad . \tag{5.6}$$

We call this metric *Efficiency* as it gives a higher score to a method that discovers surface early on.

5.5.3 Model performance on different datasets

Here we evaluate the performance of our model on data collected from different scenes as described in Sec. 5.4.5. The model was trained on the training set of *Washington2* and we report Spearman's rho as well as the loss value from Eq. Eq. (5.4) in Tab. 5.1.

The Spearman's rho shows a clear rank correlation even for the *Neighborhood* scene which features building distribution and geometry significantly different from *Washington2* which was used to generate training data. Interestingly, the model shows a high rank correlation for the *SanFrancisco* scene which features tall buildings and thus requires our model to generalize to different occupancy map distributions at high viewpoints.

Method \ Scene	Evaluation on different datasets: *W2* train	*W2* test	*W1*	*P*	*S*	*N*
Spearman's rho	0.88	0.87	0.83	0.69	0.73	0.48
Loss value	0.25	0.28	0.43	0.63	0.60	0.93

Table 5.1: Spearman's rho and loss values for our model on the different datasets. Despite the different building distribution and geometry of the test scenes (i.e. all scenes but *Washington2*) compared to training data Spearman's rho value shows a high rank correlation with the oracle score. This is even the case for the *Neighborhood* scene which features building shapes and trees unlike any in the training data. Scene names are abbreviated as *W2* for *Washington2*, *W1* for *Washington1*, *P* for *Paris*, *S* for *SanFrancisco* and *N* for *Neighborhood*.

5.5.4 Comparison with baselines

In Table 5.2 we compare the performance of our models against related hand-crafted utility functions [132, 134, 135]. Our method consistently outperforms the existing functions in terms of the efficiency measure, and as shown in Table 5.3, is faster to compute than other methods.

We also show plots of observed surface voxels vs. time for our model, the oracle with access to ground truth and baseline methods in Fig. 5.10. Note that the scenes shown have not been used to generate any training data. The results show that our method performs better compared to the baseline methods at all time steps. Additionally the behavior of our method is consistent over all scenes while the performance of the baselines varies from scene to scene. The progression of reconstructed 3D models is shown by the renderings of the occupancy map at different times.

Method \ Scene	Evaluation on different scenes				
	W2	*W1*	*P*	*S*	*N*
Frontier	0.40	0.29	0.57	0.09	0.27
AverageEntropy [134]	0.26	0.36	0.32	0.30	0.50
ProximityCount [134]	0.52	0.47	0.37	0.23	0.60
Ours	**0.91**	**0.88**	**0.87**	**0.77**	**0.74**
Oracle (GT access)	1.00	1.00	1.00	1.00	1.00

Table 5.2: Comparison of *Efficiency* metric. Our method achieves a higher value than the other utility functions on all scenes showing that our learned models can generalize to other scenes. Note that the model is trained only on data recorded from *Washington2*. *Efficiency* values are normalized with respect to the oracle for easier comparison. Scene names are abbreviated as *W2* for *Washington2*, *W1* for *Washington1*, *P* for *Paris*, *S* for *SanFrancisco* and *N* for *Neighborhood*.

	Computation time per step			
	Frontier	ProximityCount	AverageEntropy	Ours
Time in s	0.61	5.89	8.35	**0.57**

Table 5.3: Comparison of computation time per step. Our method is as fast as a simple raycast in the *Frontier* method and more than 10× faster than *ProximityCount* and *AverageEntropy*.

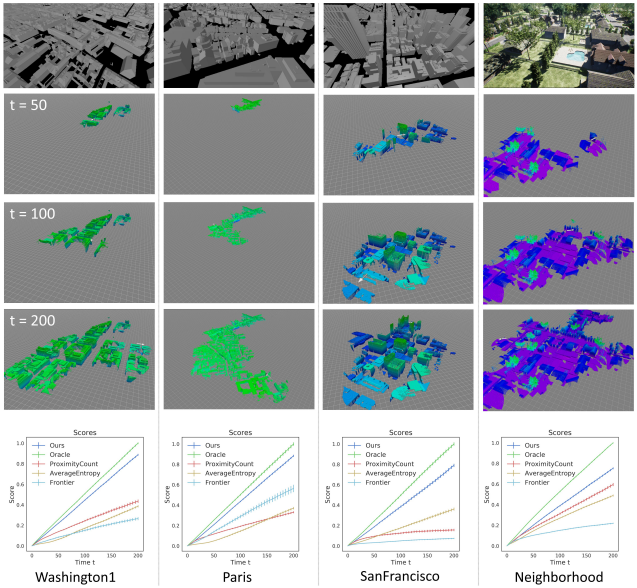

Figure 5.10: Results on all test scenes. Top row: Visualization of the underlying mesh model. Row 2-4: Reconstructed 3D models at different time steps. Shown are only occupied voxels and the color coding indicates the voxel position along the z-axis. Bottom row: Plot of observed surface voxels vs. time for all methods, the oracle with access to ground truth and the baseline methods. Our method performs the best and approaches the oracle's performance. Best viewed in color and zoomed in.

5.5.5 Noisy input sensor

While all our training is done on simulated data using ground truth depth images our intermediate state representation as an occupancy

Evaluation using noisy depth images (normalized)						
Noise	none	low	medium	high	very high	stereo
eff	1.00	0.99	1.01	0.99	1.02	0.99

Table 5.4: Comparison of our method using noisy depth images. *Efficiency* values are normalized to the noise-free case. For the noise cases 40% of pixels in each depth image were dropped and each remaining pixel was perturbed by normal noise ($\sigma = 0.1m$ for low, $\sigma = 0.2m$ for medium, $\sigma = 0.5m$ for high, $\sigma = 1.0m$ for very high). In the case of stereo matching we used a photo realistic rendering engine to generate stereo images with a baseline of $0.5m$. A disparity and depth image was computed using semi global matching [155]. Note that all values have a standard deviation of $\approxeq 0.03$.

map makes our models robust to the noise characteristics of the input sensor. Here we evaluate the performance of our models at test time with depth images perturbed by noise of different magnitude. Additionally we test our models with depth images computed from a virtual stereo camera. To this end we utilize a photorealistic game engine to synthesize RGB stereo pairs and compute depth maps with semi-global matching.

Episodes were run with noisy depth images and the viewpoint sequence was recorded. We replayed the same viewpoint sequences and used ground truth depth images to build up an occupancy map and measure the efficiency. Resulting *Efficiency* values are listed in Table 5.4. One can see that our method is robust to different noise levels. More importantly, even with depth images from the virtual stereo camera, resulting in realistic perturbations of the depth images , our method does not degrade.

	Evaluation on additional real data			
	Frontier	ProximityCount [134]	**Ours**	Oracle (GT access)
Outdoor	0.46	0.58	**0.90**	1.00
Indoor	0.44	0.52	**0.78**	1.00

Table 5.5: Comparison of *Efficiency* metric on the additional real data. Our method achieves a higher value than the other utility functions on both ourdoor and indoor scenes. Note that in both cases the model was trained on data recorded from a single scene that was different from the evaluation scene. *Efficiency* values are normalized with respect to the oracle for easier comparison.

5.5.6 Additional results on real data

To show that our method is general and also works with real scenes we conducted additional experiments on high-fidelity 3D reconstructions of buildings and on the 2D-3D-S indoor dataset [156] that was acquired with a Matterport* camera. Result are shown in Tab. 5.5, Fig. 5.11 and Fig. 5.12. For the outdoor case we trained our model on the church (Fig. 5.11a) and evaluated on the historic building (Fig. 5.11.c). Despite the differences of both buildings in terms of geometry and scale (the historic building is about $2x$ smaller in each dimension) our model is able to generalize. For the indoor case we trained on *Area1* and evaluated on *Area5b* of the 2D-3D-S indoor dataset [156]. Both experiments demonstrate that our method also works with real detailed scenes.

*https://matterport.com/

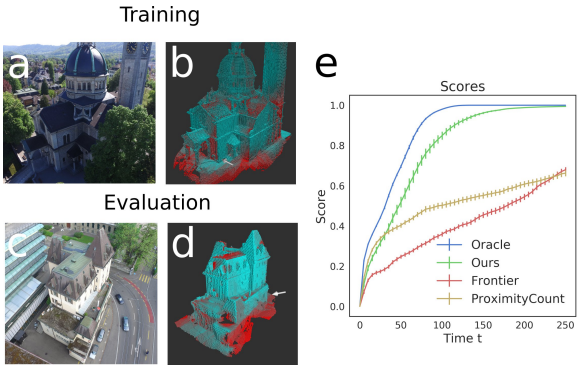

Figure 5.11: Shown are example explorations on *real* outdoor data – (a) Picture of church scene. (b) Occupancy map of the church scene (training data) (200 steps). (c) Picture of historic building scene. (d) Occupancy map of the historic building scene (evaluation) (100 steps). (e) Performance plot for the historic building scene. Color coding of observed voxels: High uncertainty (red) and low uncertainty (cyan).

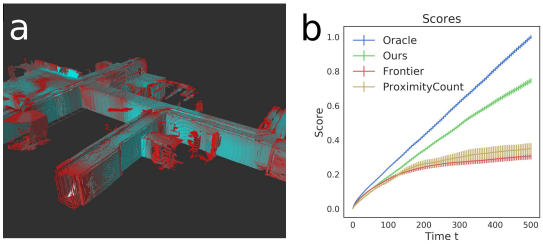

Figure 5.12: Shown are example explorations on *real* indoor data – (a) Occupancy map of S3Dis Area5b (400 steps). (b) Performance plot for S3Dis Area5b (training on Area1). Color coding of observed voxels: High uncertainty (red) and low uncertainty (cyan).

5.6 Discussion

We presented an approach for efficient exploration of unknown 3D environments by predicting the utility of new views using a 3D ConvNet. We input a novel multi-scale voxel representation of an underlying occupancy map, which represents the current model of the environment. Pairs of input and target utility score are obtained from an oracle that has access to ground truth information. Importantly, our model is able to generalize to scenes other than the training data and the underlying occupancy map enables robustness to noisy sensor input such as depth images from a stereo camera. Experiments indicate that our approach improves upon previous methods in terms of reconstruction efficiency.

Limitations of our method include dependence on the surface voxel distribution in the training data. In future work, it would be interesting to see how the method performs on vastly different geometries such as rock formations and other natural landscapes. Similarly, our model is bound to the map resolution and mapping parameters used in the training data.

Another limitation is the underlying assumption on a static scene. A dynamic object such as a human walking in front of the camera would lead to occupied voxels that do not correspond to a static object. While these voxels can change their state from occupied to free after additional observations if the human walked away the intermediate occupancy map can lead to utility predictions that are not desired. A possible solution to address this problem is to identify and segment dynamic objects in the depth maps before integrating them into the occupancy map.

Chapter 6

Conclusion

In this thesis we described the design of planning systems for quadrotors in various use cases. While being targeted to specific use cases these case studies also provide a reference for the development of other autonomous systems in related areas. We conclude this thesis by giving a discussion on the previous chapters and clarifying the contributions.

In Chapter 3 we introduced a trajectory generation framework providing smooth and pleasant camera motion for a camera-equipped quadrotor. In this setting we are given a high-level description of the quadrotor flight in terms of keyframes with desired positions or look-at targets at specific times. We used direct collocation to transcribe the trajectory generation problem. Without additional non-linear terms and constraints the resulting optimization problem is a quadratic program and can be solved efficiently. However, our framework is flexible and allows the integration of additional non-linear costs and constraints, typically at the cost of increased computation time. For this case we introduced a sequential quadratic programming scheme and we demonstrated the flexibility of our framework on different use cases. Our framework can readily be adapted and extended

for other application scenarios.

Following this, in Chapter 4 we developed a system to generate flight plans allowing efficient image capture for high-quality 3D reconstructions in urban and crowded settings. In this setting we are given a prior coarse scan of the region of interest and want to compute a set of viewpoints for a final high-quality 3D reconstruction. In addition to the task of selecting a suitable set of viewpoints we are constrained by the battery time of the quadrotor and the necessity of only performing collision-free motion. To solve the problem we introduced a *submodular* viewpoint utility that takes occlusions by occupied or unknown space into account. We handled collision-free motion using sampling-based planning with fast collision checks using the prior occupancy map. We then formulated the task as a *submodular orienteering* problem. and proposed an approximate solver that performs favorable in comparison to other approaches. The resulting 3D models demonstrated very high surface detail and we showed that quantitatively our method also compares favorably to other approaches. Individual components of our system such as the *submodular* viewpoint utility, our planning strategy for free-space motion and our solver for *submodular orienteering* problems can be used independently in other systems.

Finally, in Chapter 5 we tackled the problem of autonomously acquiring initial coarse 3D models of unknown scenes. In this setting the quadrotor iterates between moving to new viewpoints and integrating measurements into an occupancy map. State-of-the-art methods compute a heuristic viewpoint utility based on ray-casts into the occupancy map to decide on the next viewpoint to visit. We proposed to learn the viewpoint utility and formulated the next-best-view strategy as an imitation learning problem. By introducing a viewpoint-aligned, multi-scale and volumetric representation we were able to harness the expressive power of 3D convolutional neural networks and showed that our trained models significantly outperform the state-of-art. Furthermore, our models can generalize to scenes with considerably different geometry than the training data. Our demonstrated approach of finding utility functions by posing the task as an imitation learning prob-

lem of an oracle policy can be adapted to other use cases whenever an expert policy can be defined on the ground truth data that is not available at evaluation time.

6.1 Future Work

Despite much work and progress in the area of planning for autonomous systems the design of such systems is still a difficult endeavor. While solutions exist for individual blocks, such as low-level control, global path planning, state estimation, object perception and mapping, they often make too specific assumptions or can fail in corner cases. When putting these components together in a closed-loop system the emergent behavior can lead to unexpected situations and failure of these components.

General Directions We see two key challenges that remain a fruitful area for further research in the domain of autonomous systems design:

- In the case of dynamic environments with multiple independent agents the system needs to plan and reason over uncertain and coarse predictions of the other agents. The challenge is to model the other agents at a level of abstraction that allows fast computation and captures enough modalities of the other agents behavior to make robust and safe decisions.
- Much work in the planning domain does not focus on the perception component with the implicit assumption that it will be almost perfectly solved at some point in the future. We feel that this might never happen. Instead, both parts should be considered jointly by reasoning over uncertainties and occlusions from the perception component.

Apart from these general direction we identified specific promising areas of future research that we believe are worth undertaking. Here we give an overview of these avenues.

Trajectory Planning In terms of trajectory planning there still exists a disconnect between global planning and local planning. One approach for global planning is building a graph over a regular state lattice or other state decompositions such as visibility graphs [157]. A plan or path is then computed by performing a search in the graph. These graphs usually contain no kinematics or dynamics to make the computation feasible. The path is then passed to a local planner that incorporates kinematic and dynamic constraints and uses the global path as a *warm start* for the optimization. The disconnect manifests itself when the global planner finds a path that turns out to not be feasible in the local planner. While kinematics and dynamics can be integrated into sampling based approaches such as the family of RRT methods [5, 6, 7] the level of detail of the dynamic and kinematic constraints has a strong impact on the computational cost. In many cases it is infeasible to incorporate a full model and again the problem is split into a global and local planning approach. What is missing is a feedback loop from the local planner to the global planner to inform about feasible paths. However, it is not enough to simply invalidate a single path but the whole homotopy of the path has to be invalidated to prevent unnecessary local planning of similar paths. It is not directly clear how this can be handled in a sampling based planner but a promising approach could be to backtrack along the invalidated path of the tree and check if all children still belong to the same homotopy. This check could be approximated by testing for obstacles between the children. We can perform this backtracking until we find a node that leads to other homotopies and prune the branch with the homotopy that was invalidated.

Trajectory Planning for Multi-View Stereo Reconstruction In terms of planning for Multi-View Stereo reconstruction we only touched the potential of performing iterative reconstructions. In this context a very fruitful direction would be to learn a good metric that informs whether stereo matching of two views would result in good depth maps. We conjecture that using the relative pose and both images

as features would allow to learn such a metric and there might be an opportunity to use existing reconstructions from datasets. Even if only sparse reconstructions are available many stereo matched and triangulated pixels could be invalidated based on the sparse geometry. Another interesting direction is the use of a fast online SfM and MVS system to enable on-the-fly computation of the next best view.

Learning a Viewpoint Utility Score Our work on learning utility functions for exploration also suggests several directions for future work. We used our learned utility function to implement a greedy next-best-view algorithm. However, our utility function could be used to develop more sophisticated policies that look multiple steps ahead. In addition, our approach could be extended to be used in a generative way to predict future states of the 3D occupancy map or to predict 2D depth maps for unobserved views. This could be used for model completion or hole-filling which has numerous applications in computer vision and robotics.

Bibliography

[1] Krause, A., Golovin, D.: Submodular function maximization. Tractability: Practical Approaches to Hard Problems **3** (2012) 8

[2] Betts, J.T.: Practical methods for optimal control and estimation using nonlinear programming. Volume 19. Siam (2010)

[3] Kelly, M.: An introduction to trajectory optimization: how to do your own direct collocation. SIAM Review **59** (2017) 849–904

[4] Deits, R., Tedrake, R.: Footstep planning on uneven terrain with mixed-integer convex optimization. In: Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on, IEEE (2014) 279–286

[5] LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. (1998)

[6] Karaman, S., Frazzoli, E.: Incremental sampling-based algorithms for optimal motion planning. Robotics Science and Systems VI **104** (2010) 2

[7] Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. The international journal of robotics research **30** (2011) 846–894

[8] Kuffner, J.J., LaValle, S.M.: Rrt-connect: An efficient approach to single-query path planning. In: Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on. Volume 2., IEEE (2000) 995–1001

[9] Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C., Burgard, W.: OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots (2013) Software available at `http://octomap.github.com`.

[10] Kazhdan, M., Hoppe, H.: Screened poisson surface reconstruction. ACM Transactions on Graphics (TOG) **32** (2013) 29

[11] Hartley, R., Zisserman, A.: Multiple view geometry in computer vision. Cambridge university press (2003)

[12] Szeliski, R.: Computer vision: algorithms and applications. Springer Science & Business Media (2010)

[13] Fuhrmann, S., Langguth, F., Moehrle, N., Waechter, M., Goesele, M.: Mve—an image-based reconstruction environment. Computers & Graphics **53** (2015) 44–53

[14] Schönberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2016)

[15] Furukawa, Y., Ponce, J.: Accurate, dense, and robust multiview stereopsis. IEEE transactions on pattern analysis and machine intelligence **32** (2010) 1362–1376

[16] Furukawa, Y., Hernández, C., et al.: Multi-view stereo: A tutorial. Foundations and Trends® in Computer Graphics and Vision **9** (2015) 1–148

[17] Galliani, S., Lasinger, K., Schindler, K.: Massively parallel multiview stereopsis by surface normal diffusion. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 873–881

[18] Schönberger, J.L., Zheng, E., Pollefeys, M., Frahm, J.M.: Pixelwise view selection for unstructured multi-view stereo. In: European Conference on Computer Vision (ECCV). (2016)

[19] Hirschmuller, H., Scharstein, D.: Evaluation of cost functions for stereo matching. In: Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, IEEE (2007) 1–8

[20] Zbontar, J., LeCun, Y.: Computing the stereo matching cost with a convolutional neural network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2015) 1592–1599

[21] Hornung, A., Zeng, B., Kobbelt, L.: Image selection for improved multi-view stereo. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE (2008) 1–8

[22] Bailer, C., Finckh, M., Lensch, H.P.: Scale robust multi view stereo. In: European Conference on Computer Vision, Springer (2012) 398–411

[23] Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: Patchmatch: A randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics (ToG) **28** (2009) 24

[24] Zheng, E., Dunn, E., Jojic, V., Frahm, J.M.: Patchmatch based joint view selection and depthmap estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2014) 1510–1517

[25] Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions—i. Mathematical programming **14** (1978) 265–294

[26] Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: A survey. European Journal of Operational Research **209** (2011) 1–10

[27] Chekuri, C., Pal, M.: A recursive greedy algorithm for walks in directed graphs. In: Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on, IEEE (2005) 245–253

[28] Singh, A., Krause, A., Guestrin, C., Kaiser, W., Batalin, M.: Efficient Planning of Informative Paths for Multiple Robots. In: IJCAI. (2007)

[29] Zhang, H., Vorobeychik, Y.: Submodular optimization with routing constraints. In: AAAI. (2016) 819–826

[30] Tschiatschek, S., Singla, A., Krause, A.: Selecting sequences of items via submodular maximization. In: AAAI. (2017) 2667–2673

[31] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) `http://www.deeplearningbook.org`.

[32] Gebhardt, C., Hepp, B., Nägeli, T., Stevšić, S., Hilliges, O.: Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, ACM (2016) 2508–2519

[33] Higuchi, K., Shimada, T., Rekimoto, J.: Flying Sports Assistant: External Visual Imagery Representation for Sports Training. In: Augmented Human International Conference (AH '11), ACM (2011) 7:1—-7:4

[34] Mueller, F.F., Muirhead, M.: Jogging with a quadcopter. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. CHI '15, New York, NY, USA, ACM (2015) 2023–2032

[35] Nitta, K., Higuchi, K., Rekimoto, J.: HoverBall: Augmented Sports with a Flying Ball. In: Augmented Human International

Conference (AH '14). AH '14, New York, NY, USA, ACM (2014) 13:1—-13:4

[36] Scheible, J., Hoth, A., Saal, J., Su, H.: Displaydrone: a flying robot based interactive display. In: ACM International Symposium on Pervasive Displays (PerDis '13), New York, New York, USA, ACM Press (2013) 49

[37] ArsElectronica: Spaxels (2012) `http://www.aec.at/spaxels/`.

[38] Cannes Festival: New Directors' Showcase. Video (2012)

[39] Mascelli, J.V.: The five C's of cinematography: motion picture filming techniques. Silman-James Press (1998)

[40] Diaz, T.: Lights, drone... action. Spectrum, IEEE **52** (2015) 36–41

[41] Naseer, T., Sturm, J., Cremers, D.: Followme: Person following and gesture recognition with a quadrocopter. Proc. IROS (2013)

[42] Higuchi, K., Rekimoto, J.: Flying Head: Head-synchronized Unmanned Aerial Vehicle Control for Flying Telepresence. In: SIGGRAPH Asia 2012 E-Tech. SA '12, New York, NY, USA, ACM (2012) 12:1—-12:2

[43] Gleicher, M.L., Liu, F.: Re-cinematography: Improving the camerawork of casual video. ACM Trans. Multimedia Comput. Commun. Appl. **5** (2008) 2:1–2:28

[44] Grundmann, M., Kwatra, V., Essa, I.: Auto-directed video stabilization with robust l1 optimal camera paths. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. (2011) 225–232

[45] Liu, S., Yuan, L., Tan, P., Sun, J.: Bundled camera paths for video stabilization. ACM Trans. Graph. **32** (2013) 78:1–78:10

[46] Kopf, J., Cohen, M.F., Szeliski, R.: First-person hyper-lapse videos. ACM Trans. Graph. **33** (2014) 78:1–78:10

[47] Christie, M., Machap, R., Normand, J.M., Olivier, P., Pickering, J.: Virtual camera planning: A survey. In: Smart Graphics, Springer (2005) 40–52

[48] Yeh, I.C., Lin, C.H., Chien, H.J., Lee, T.Y.: Efficient camera path planning algorithm for human motion overview. Computer Animation and Virtual Worlds **22** (2011) 239–250

[49] Li, T.Y., Cheng, C.C.: Real-time camera planning for navigation in virtual environments. In Butz, A., Fisher, B., Krüger, A., Olivier, P., Christie, M., eds.: Smart Graphics. Volume 5166 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 118–129

[50] Umetani, N., Koyama, Y., Schmidt, R., Igarashi, T.: Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. ACM Trans. Graph. **33** (2014) 65:1–65:10

[51] Martin, T., Umetani, N., Bickel, B.: Omniad: Data-driven omni-directional aerodynamics. ACM Trans. Graph. **34** (2015) 113:1–113:12

[52] Yoshida, S., Shirokura, T., Sugiura, Y., Sakamoto, D., Ono, T., Inami, M., Igarashi, T.: RoboJockey: Designing an Entertainment Experience with Robots. IEEE computer graphics and applications (2015) 1

[53] Liu, K., Sakamoto, D., Inami, M., Igarashi, T.: Roboshop: Multi-layered sketching interface for robot housework assignment and management. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '11, New York, NY, USA, ACM (2011) 647–656

[54] Sakamoto, D., Honda, K., Inami, M., Igarashi, T.: Sketch and run. In: ACM SIGCHI, New York, New York, USA, ACM Press (2009) 197

[55] Sugiura, Y., Sakamoto, D., Withana, A., Inami, M., Igarashi, T.: Cooking with robots. In: ACM SIGCHI, New York, New York, USA, ACM Press (2010) 2427

[56] Kato, J., Sakamoto, D., Igarashi, T.: Phybots: A toolkit for making robotic things. In: Proceedings of the Designing Interactive Systems Conference. DIS '12, New York, NY, USA, ACM (2012) 248–257

[57] Zhao, S., Nakamura, K., Ishii, K., Igarashi, T.: Magic cards: A paper tag interface for implicit robot control. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '09, New York, NY, USA, ACM (2009) 173–182

[58] Lupashin, S., D'Andrea, R.: Adaptive fast open-loop maneuvers for quadrocopters. Autonomous Robots **33** (2012) 89–102

[59] Mellinger, D., Kumar, V.: Minimum snap trajectory generation and control for quadrotors. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE (2011) 2520–2525

[60] Joubert, N., Roberts, M., Truong, A., Berthouzoz, F., Hanrahan, P.: An interactive tool for designing quadrotor camera shots. ACM Transactions on Graphics (TOG) **34** (2015) 238

[61] Mahony, R., Kumar, V., Corke, P.: Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor. IEEE Robotics & Automation Magazine **19** (2012) 20–32

[62] Flash, T., Hogan, N.: The coordination of arm movements: an experimentally confirmed mathematical model. The journal of Neuroscience **5** (1985) 1688–1703

[63] Alonso-Mora, J., Naegeli, T., Siegwart, R., Beardsley, P.: Collision avoidance for aerial vehicles in multi-agent scenarios. Autonomous Robots **39** (2015) 101–121

[64] Meier, L., Tanskanen, P., Heng, L., Lee, G.H., Fraundorfer, F., Pollefeys, M.: PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. Autonomous Robots **33** (2012) 21–39

[65] Faiz, N., Agrawal, S.K., Murray, R.M.: Trajectory planning of differentially flat systems with dynamics and inequalities. Journal of Guidance, Control, and Dynamics **24** (2001) 219–227

[66] Lee, T., Leok, M., McClamroch, N.H.: Nonlinear robust tracking control of a quadrotor uav on se (3). Asian Journal of Control **15** (2013) 391–408

[67] Nägeli, T., Conte, C., Domahidi, A., Morari, M., Hilliges, O.: Environment-independent formation flight for micro aerial vehicles. In: Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on. (2014) 1141–1146

[68] Hepp, B., Nießner, M., Hilliges, O.: Plan3d: Viewpoint and trajectory optimization for aerial multi-view stereo reconstruction. arXiv preprint arXiv:1705.09314 (2017)

[69] Jancosek, M., Pajdla, T.: Multi-view reconstruction preserving weakly-supported surfaces. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE (2011) 3121–3128

[70] Fuhrmann, S., Langguth, F., Goesele, M.: Mve-a multi-view reconstruction environment. In: GCH. (2014) 11–18

[71] Langguth, F., Sunkavalli, K., Hadap, S., Goesele, M.: Shading-aware multi-view stereo. In: European Conference on Computer Vision, Springer (2016) 469–485

[72] Schöps, T., Schönberger, J.L., Galliani, S., Sattler, T., Schindler, K., Pollefeys, M., Geiger, A.: A multi-view stereo benchmark with high-resolution images and multi-camera videos. In: Proc. CVPR. Volume 3. (2017)

[73] Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. ACM Transactions on Graphics (2017)

[74] Seitz, S.M., Curless, B., Diebel, J., Scharstein, D., Szeliski, R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In: Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on. Volume 1., IEEE (2006) 519–528

[75] Goesele, M., Snavely, N., Curless, B., Hoppe, H., Seitz, S.M.: Multi-view stereo for community photo collections. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, IEEE (2007) 1–8

[76] Pix4D: Pix4d. `https://pix4d.com/` (2017)

[77] Agisoft, L.: Agisoft photoscan user manual: Professional edition (2014)

[78] Chen, S., Li, Y., Kwok, N.M.: Active vision in robotic systems: A survey of recent developments. The International Journal of Robotics Research **30** (2011) 1343–1377

[79] Hepp, B.: Plan3d open source code. `https://github.com/bennihepp/Quad3DR` (2018)

[80] Roberts, M., Truong, A., Dey, D., Sinha, S., Kapoor, A., Joshi, N., Hanrahan, P.: Submodular trajectory optimization for aerial 3d scanning. arXiv preprint arXiv:1705.00703 (2017)

[81] Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle adjustment—a modern synthesis. In: International workshop on vision algorithms, Springer (1999) 298–372

[82] Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: exploring photo collections in 3d. In: ACM transactions on graphics (TOG). Volume 25., ACM (2006) 835–846

[83] Snavely, N., Seitz, S.M., Szeliski, R.: Modeling the world from internet photo collections. International journal of computer vision **80** (2008) 189–210

[84] Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building rome in a day. In: 2009 IEEE 12th international conference on computer vision, IEEE (2009) 72–79

[85] Agarwal, S., Snavely, N., Seitz, S., Szeliski, R.: Bundle adjustment in the large. Computer Vision–ECCV 2010 (2010) 29–42

[86] Wu, C., Agarwal, S., Curless, B., Seitz, S.M.: Multicore bundle adjustment. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE (2011) 3057–3064

[87] Goesele, M., Curless, B., Seitz, S.M.: Multi-view stereo revisited. In: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on. Volume 2., IEEE (2006) 2402–2409

[88] Strecha, C., Von Hansen, W., Van Gool, L., Fua, P., Thoennessen, U.: On benchmarking camera calibration and multi-view stereo for high resolution imagery. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, Ieee (2008) 1–8

[89] Fuhrmann, S., Goesele, M.: Floating scale surface reconstruction. ACM Transactions on Graphics (TOG) **33** (2014) 46

[90] Farid, H., Lee, S., Bajcsy, R.: View selection strategies for multi-view, wide-base stereo. Technical report, Technical Report MS-CIS-94-18, University of Pennsylvania (1994)

[91] Kutulakos, K.N., Dyer, C.R.: Recovering shape by purposive viewpoint adjustment. In: Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on, IEEE (1992) 16–22

[92] Vázquez, P.P., Feixas, M., Sbert, M., Heidrich, W.: Automatic view selection using viewpoint entropy and its application to image-based modelling. In: Computer Graphics Forum. Volume 22., Wiley Online Library (2003) 689–700

[93] Haner, S., Heyden, A.: Covariance propagation and next best view planning for 3d reconstruction. In: Computer Vision–ECCV 2012. Springer (2012) 545–556

[94] Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM (1996) 303–312

[95] Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohi, P., Shotton, J., Hodges, S., Fitzgibbon, A.: Kinectfusion: Real-time dense surface mapping and tracking. In: Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on, IEEE (2011) 127–136

[96] Nießner, M., Zollhöfer, M., Izadi, S., Stamminger, M.: Real-time 3d reconstruction at scale using voxel hashing. ACM Transactions on Graphics (TOG) **32** (2013) 169

[97] Chen, J., Bautembach, D., Izadi, S.: Scalable real-time volumetric surface reconstruction. ACM Transactions on Graphics (TOG) **32** (2013) 113

[98] Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., Theobalt, C.: Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. arXiv preprint arXiv:1604.01093 (2016)

[99] Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: Proceedings of the fourth Eurographics symposium on Geometry processing. Volume 7. (2006)

[100] Robotics, D.: 3dr site scan. `https://3dr.com/` (2017)

[101] Heng, L., Lee, G.H., Fraundorfer, F., Pollefeys, M.: Real-time photo-realistic 3d mapping for micro aerial vehicles. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. (2011) 4012–4019

[102] Loianno, G., Thomas, J., Kumar, V.: Cooperative localization and mapping of mavs using rgb-d sensors. In: Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE (2015) 4021–4028

[103] Sturm, J., Bylow, E., Kerl, C., Kahl, F., Cremer, D.: Dense tracking and mapping with a quadrocopter. Unmanned Aerial Vehicle in Geomatics (UAV-g), Rostock, Germany (2013)

[104] Du, J., Mouser, C., Sheng, W.: Design and evaluation of a teleoperated robotic 3-d mapping system using an rgb-d sensor. IEEE Transactions on Systems, Man, and Cybernetics: Systems **46** (2016) 718–724

[105] Scaramuzza, D., Achtelik, M.C., Doitsidis, L., Friedrich, F., Kosmatopoulos, E., Martinelli, A., Achtelik, M.W., Chli, M., Chatzichristofis, S., Kneip, L., et al.: Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in gps-denied environments. IEEE Robotics & Automation Magazine **21** (2014) 26–40

[106] Yamauchi, B.: A frontier-based approach for autonomous exploration. In: Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on, IEEE (1997) 146–151

[107] Fraundorfer, F., Heng, L., Honegger, D., Lee, G.H., Meier, L., Tanskanen, P., Pollefeys, M.: Vision-based autonomous mapping and exploration using a quadrotor mav. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE (2012) 4557–4564

[108] Shen, S., Michael, N., Kumar, V.: Autonomous multi-floor indoor navigation with a computationally constrained mav. In: Robotics and automation (ICRA), 2011 IEEE international conference on, IEEE (2011) 20–25

[109] Hoppe, C., Klopschitz, M., Rumpler, M., Wendel, A., Kluckner, S., Bischof, H., Reitmayr, G.: Online feedback for structure-from-motion image acquisition. In: BMVC. Volume 2. (2012) 6

[110] Kriegel, S., Rink, C., Bodenmüller, T., Suppa, M.: Efficient next-best-scan planning for autonomous 3d surface reconstruction of unknown objects. Journal of Real-Time Image Processing **10** (2015) 611–631

[111] Wenhardt, S., Deutsch, B., Angelopoulou, E., Niemann, H.: Active visual object reconstruction using d-, e-, and t-optimal next best views. In: 2007 IEEE Conference on Computer Vision and Pattern Recognition, IEEE (2007) 1–7

[112] Forster, C., Pizzoli, M., Scaramuzza, D.: Appearance-based active, monocular, dense reconstruction for micro aerial vehicles. In: Robotics: Science and Systems (RSS). (2014)

[113] Dunn, E., Frahm, J.M.: Next best view planning for active model improvement. In: BMVC. (2009) 1–11

[114] Mendez, O., Hadfield, S., Pugeault, N., Bowden, R.: Taking the scenic route to 3d: Optimising reconstruction from moving cameras. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017) 4677–4685

[115] Khalfaoui, S., Seulin, R., Fougerolle, Y., Fofi, D.: An efficient method for fully automatic 3d digitization of unknown objects. Computers in Industry **64** (2013) 1152–1160

[116] Wu, S., Sun, W., Long, P., Huang, H., Cohen-Or, D., Gong, M., Deussen, O., Chen, B.: Quality-driven poisson-guided autoscanning. ACM Transactions on Graphics **33** (2014)

[117] Fan, X., Zhang, L., Brown, B., Rusinkiewicz, S.: Automated view and path planning for scalable multi-object 3d scanning. ACM Transactions on Graphics (TOG) **35** (2016) 239

[118] Xu, K., Huang, H., Shi, Y., Li, H., Long, P., Caichen, J., Sun, W., Chen, B.: Autoscanning for coupled scene reconstruction and proactive object analysis. ACM Trans. Graph. **34** (2015) 177:1–177:14

[119] Fraser, C.: Network design considerations for non-topographic photogrammetry. Photogrammetric Engineering and Remote Sensing **50** (1984) 1115–1126

[120] Mason, S., et al.: Heuristic reasoning strategy for automated sensor placement. Photogrammetric engineering and remote sensing **63** (1997) 1093–1101

[121] Olague, G., Mohr, R.: Optimal camera placement for accurate reconstruction. Pattern Recognition **35** (2002) 927–944

[122] Hoppe, C., Wendel, A., Zollmann, S., Pirker, K., Irschara, A., Bischof, H., Kluckner, S.: Photogrammetric camera network design for micro aerial vehicles. In: Computer vision winter workshop (CVWW). Volume 8. (2012) 1–3

[123] Bircher, A., Alexis, K., Burri, M., Oettershagen, P., Omari, S., Mantel, T., Siegwart, R.: Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. In: Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE (2015) 6423–6430

[124] Huang, R., Zou, D., Vaughan, R., Tan, P.: Active image-based modeling. arXiv preprint arXiv:1705.01010 (2017)

[125] Waechter, M., Moehrle, N., Goesele, M.: Let there be color! large-scale texturing of 3d reconstructions. In: European Conference on Computer Vision, Springer (2014) 836–850

[126] Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica **2** (1982) 385–393

[127] Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. MIT press (2005)

[128] Coulter, R.C.: Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST (1992)

[129] Hepp, B., Dey, D., Sinha, S.N., Kapoor, A., Joshi, N., Hilliges, O.: Learn-to-score: Efficient 3d scene exploration by predicting view utility. In: The European Conference on Computer Vision (ECCV). (2018)

[130] Xu, K., Zheng, L., Yan, Z., Yan, G., Zhang, E., Nießner, M., Deussen, O., Cohen-Or, D., Huang, H.: Autonomous reconstruction of unknown indoor scenes guided by time-varying tensor fields. ACM Transactions on Graphics 2017 (TOG) (2017)

[131] Roberts, M., Dey, D., Truong, A., Sinha, S., Shah, S., Kapoor, A., Hanrahan, P., Joshi, N.: Submodular trajectory optimization for aerial 3d scanning. In: International Conference on Computer Vision (ICCV) 2017. (2017)

[132] Vasquez-Gomez, J.I., Sucar, L.E., Murrieta-Cid, R., Lopez-Damian, E.: Volumetric next-best-view planning for 3d object reconstruction with positioning error. International Journal of Advanced Robotic Systems **11** (2014) 159

[133] Heng, L., Gotovos, A., Krause, A., Pollefeys, M.: Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In: ICRA. (2015)

[134] Isler, S., Sabzevari, R., Delmerico, J., Scaramuzza, D.: An information gain formulation for active volumetric 3d reconstruction. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), IEEE (2016) 3477–3484

[135] Delmerico, J., Isler, S., Sabzevari, R., Scaramuzza, D.: A comparison of volumetric information gain metrics for active 3d object reconstruction. Autonomous Robots (2017) 1–12

[136] Feige, U.: A threshold of ln n for approximating set cover. JACM (1998)

[137] Golovin, D., Krause, A.: Adaptive submodularity: Theory and applications in active learning and stochastic optimization. JAIR (2011)

[138] Hollinger, G.A., Englot, B., Hover, F.S., Mitra, U., Sukhatme, G.S.: Active planning for underwater inspection and the benefit of adaptivity. IJRR (2012)

[139] Choudhury, S., Kapoor, A., Ranade, G., Scherer, S., Dey, D.: Adaptive information gathering via imitation learning. Robotics Science and Systems (2017)

[140] Devrim Kaba, M., Gokhan Uzunbas, M., Nam Lim, S.: A reinforcement learning approach to the view planning problem. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017) 6933–6941

[141] Song, S., Xiao, J.: Deep sliding shapes for amodal 3d object detection in rgb-d images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 808–816

[142] Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: Scannet: Richly-annotated 3d reconstructions of indoor scenes. http://arxiv.org/abs/1702.04405 (2017)

[143] Riegler, G., Ulusoy, A.O., Geiger, A.: Octnet: Learning deep 3d representations at high resolutions. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2017)

[144] Dai, A., Qi, C.R., Nießner, M.: Shape completion using 3d-encoder-predictor cnns and shape synthesis. http://arxiv.org/abs/1612.00101 (2016)

[145] Ge, L., Liang, H., Yuan, J., Thalmann, D.: 3d convolutional neural networks for efficient and robust hand pose estimation from single depth images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017) 1991–2000

[146] Zeng, A., Song, S., Nießner, M., Fisher, M., Xiao, J., Funkhouser, T.: 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In: CVPR. (2017)

[147] Choy, C.B., Xu, D., Gwak, J., Chen, K., Savarese, S.: 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In: European Conference on Computer Vision, Springer (2016) 628–644

[148] Liu, F., Shen, C., Lin, G.: Deep convolutional neural fields for depth estimation from a single image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 5162–5170

[149] Zamir, A.R., Wekel, T., Agrawal, P., Wei, C., Malik, J., Savarese, S.: Generic 3d representation via pose estimation and matching. In: European Conference on Computer Vision, Springer (2016) 535–553

[150] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2016)

[151] He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European Conference on Computer Vision, Springer (2016) 630–645

[152] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[153] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. (2010) 249–256

[154] Bircher, A., Kamel, M., Alexis, K., Oleynikova, H., Siegwart, R.: Receding horizon" next-best-view" planner for 3d exploration. In: Robotics and Automation (ICRA), 2016 IEEE International Conference on, IEEE (2016) 1462–1468

[155] Hirschmuller, H.: Stereo processing by semiglobal matching and mutual information. IEEE Transactions on pattern analysis and machine intelligence **30** (2008) 328–341

[156] Armeni, I., Sax, S., Zamir, A.R., Savarese, S.: Joint 2d-3d-semantic data for indoor scene understanding. arXiv preprint arXiv:1702.01105 (2017)

[157] LaValle, S.M.: Planning algorithms. Cambridge university press (2006)