DISS. ETH NO. 25791

# Mapping and Planning for Safe Collision Avoidance On-board Micro-Aerial Vehicles

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)


presented by

Elena Oleynikova

MSc. ETH RSC

born on February 2, 1990
citizen of the United States of America


accepted on the recommendation of

Prof. Dr. Roland Siegwart, Examiner
Prof. Dr. Shaojie Shen, Co-examiner

2019

Autonomous Systems Lab
Department of Mechanical and Process Engineering
ETH Zurich
Switzerland

# Abstract

There is a rapidly-growing list of commercial, industrial, and humanitarian applications for Micro-Aerial Vehicles (MAVs), due to their agility, ability to move freely in 3D space, and relatively low cost. However, until recently, these vehicles were used in one of two modes: either complete manual flight with an experienced pilot, or remote inspection by following GPS waypoints. Allowing these robots to assist human operators in search and rescue (S&R) and industrial inspection scenarios requires tackling far more challenges than these two modes of operation allow.

These two environments pose specific problems that must be addressed before an MAV can be a useful collaborator. First, these areas are usually entirely or partially GPS-denied, so all state estimation must rely entirely on on-board sensing. Second, especially for search and rescue or more complex industrial inspection tasks such as boilers, communication to a ground station may not always be available and cannot be guaranteed. Therefore, all processing must be performed on-board and the robot must behave intelligently when no external commands are sent. Similarly, we can not rely on having a skilled pilot, therefore all low-level operation must be handled by the vehicle itself. Finally, the MAV must be able to safely operate very close to structure and in cluttered environments, and be able to return with a usable map for the human operators.

We propose that to create a system that is able to address all of these requirements, mapping and planning must be closely coupled. While previous work exists solving lower-level problems such as on-board state estimation and accurate control, this thesis attempts to fill the gap one level above: representing the environment and making short- and long-term plans through it. Our approach focuses on a novel map representation, specifically designed for the needs of planning and inspection tasks, and using planning algorithms that exploit all available information in the map.

We describe the specific contributions of the thesis below. First, we propose a fast local replanning method which uses local trajectory optimization on polynomial splines to push dynamically-feasible trajectories out of collision. This method allows us to do conservative planning, meaning that unknown space is considered occupied, and replan at up to 25 Hz. However, since it relies on obstacle gradients, it needs a map representation capable of storing and updating distances to obstacles incrementally.

To address this need, we design a flexible map representation called *voxblox*, which is based on dense voxel grids storing signed distance fields. We propose a method to incrementally update Euclidean Signed Distance Fields (ESDFs) out of Truncated Signed Distance Fields (TSDFs), which are traditionally used for 3D

reconstruction. We further extend TSDF-based methods to be more suitable for large voxel sizes, which are necessary for real-time planning applications. This mapping approach allows us to create high-fidelity colored meshes for assisting human operators, while simultaneously maintaining a map representation that gives collision checks in a single look-up and gradient information toward obstacles, and outperforming existing methods in computation speed.

Next, we tackle issues that our planning approach faces in extremely cluttered environments: that of falling into local minima when no path toward the goal is visible. We propose to use a local exploration strategy, which selects next viewpoints that trade-off maximizing exploration gain and minimizing distance toward the goal whenever the local planning fails. We show that this substantially increases our success rates even in environments that are up to 50% obstacles, and outperforms the standard method of using an optimistic (treating unknown space as free) global planner.

To further take advantage of information available in our map representation, we propose a deterministic topology-based planning method. We exploit the property that ridges in the ESDF form faces in the Generalized Voronoi Diagram (GVD), and from this we can extract maximum-clearance edges through the free space. We then fit a sparse graph to these edges, creating a very compact representation of the environment, which can be used to plan an order of magnitude faster than the fastest sampling-based approach, while implicitly handling narrow gaps that sampling-based methods struggle with.

Finally, we present a complete system using all components proposed in this thesis, and benchmark them on real search and rescue and industrial inspection scenarios. We show that our approach is specifically suitable for these applications and outperforms standard approaches.

# Zusammenfassung

Mikro-Luft-Fahrzeuge (MAVs) haben ein wachsendes Anwendungsgebiet für kommerzielle, industrielle und humanitäre Einsätz, aufgrund ihrer Agilität, Bewegungsfreiheit im 3D-Raum und relativ niedriger Kosten. Bisher werden diese Fluggeräte jedoch in einem von zwei Modi eingesetzt: entweder im manuellen Flug durch erfahrene Piloten oder in der Ferninspektion mit Hilfe von GPS-Wegpunkten. Damit diese Roboter menschliche Bediener bei Such- und Rettungseinsätzen (S&R) und bei der industriellen Inspektion unterstützen können, müssen weitaus mehr Herausforderungen bewältigt werden, als durch bisherigen Lösungen abgedeckt wird.

Inbesondere diese beiden Anwendungsgebiete sind problematisch. Erstens haben diese Szenarien in der Regel nur teilweise bis gar kein GPS-Abdeckung, sodass alle Zustandsschätzungen vollständig auf On-Board-Sensing basieren müssen. Zweitens, insbesondere für Such- und Rettungseinsätze oder bei komplexeren industriellen Inspektionen wie bei der Wartung von Kesseln, ist die Kommunikation mit einer Bodenstation nicht immer verfügbar. Daher muss die gesamte Verarbeitung an Bord durchgeführt werden und der Roboter muss sich intelligent verhalten, wenn keine externen Befehle gesendet werden. Ebenso können wir uns nicht auf einen qualifizierten Piloten verlassen, weshalb der gesamte Low-Level-Betrieb vom MAV selbst durchgeführt werden muss. Schliesslich muss ein MAV in der Lage sein sehr nahe an Hindernissen und in unstrukturierten Umgebungen sicher zu arbeiten und mit nützlichen Daten, z.B. einer 3D Karte für den menschlichen Bediener zurückzukehren.

Für ein geeignetes MAV-basiertes System, welches diese Anforderungen erfüllt, müssen autonome Kartierung und Pfadplanung Hand in Hand gehen. Der Grossteil der Literatur beschäftigt sich mit den nötigen Einzelfunktionen, wie z.B. on-board Zustandsschätzung und genauer Regelung der Systeme.

Diese Arbeit beschäftigt sich mit den weiter gefassten und integrativen Aufgaben der Kartierung und dem Planen kurzer bzw. langer Pfade. Unser Ansatz basiert auf einer neuartigen Kartendarstellung, die speziell auf die Bedürfnisse von Planungs- und Inspektionsaufgaben zugeschnitten ist und Planungsalgorithmen verwendet, die alle verfügbaren Informationen dieser Karten nutzen. Im Folgenden beschreiben wir die spezifischen Beiträge dieser Dissertation.

Zuerst schlagen wir eine schnelle lokale Umplanungsmethode vor, welche lokale Trajektorienoptimierung an Polynom-Splines nutzt, um dynamisch zulässige Trajektorien kollisionsfrei zu realisieren. Diese Methode ermöglicht es uns, konservativ zu planen, d.h. unbekannter Raum wird als belegt betrachtet und mit bis zu 25 Hz umgeplant. Da die Methode jedoch auf Hindernisgradienten basiert, benötigt sie eine Kartendarstellung, die in der Lage ist, Entfernungen zu Hindernissen inkre-

mentell zu speichern und zu aktualisieren.

Um diesem Bedürfnis gerecht zu werden, entwerfen wir eine flexible Kartenrepräsentation namens voxblox, die auf dichten Voxel-Gittern basiert, die *Signed Distance Fields* (SDFs) speichern. Wir schlagen eine Methode zur inkrementellen Aktualisierung von *Euclidian Signed Distance Fields* (ESDFs) aus *Truncated Signed Distance Fields* (TSDFs) vor, die traditionell für die 3D-Rekonstruktion verwendet werden. Wir erweitern TSDF-basierte Methoden, um sie besser für grosse Voxelgrössen geeignet zu machen, die für Echtzeit-Planungsanwendungen notwendig sind. Dieser Planungsansatz ermöglicht es uns, hochpräzise colorierte Meshes zur Unterstützung menschlicher Bediener zu erstellen, während wir gleichzeitig eine Kartendarstellung beibehalten, die für Kollisionskontrollen in einem einzigen Look-up Gradienteninformationen zu Hindernissen liefert und geringere Rechenzeiten als bisherige Methoden hat. Als nächstes adressieren wir die Probleme, mit denen diese beiden Methoden in besonders ungeordneten Umgebungen angewendet werden: Lokale Minima, wenn kein Weg zum Ziel sichtbar ist. Wir schlagen vor, eine lokale Explorationsstrategie zu verwenden, die die nächsten Positionen auswählt, die den Explorationsgewinn maximieren und die Entfernung zum Ziel minimieren, wenn die lokale Planung scheitert. Wir zeigen, dass dies unsere Erfolgsraten auch in Umgebungen, die bis zu 50

Um die in unserer Kartendarstellung verfügbaren Informationen weiter zu nutzen, schlagen wir eine deterministische topologiebasierte Planungsmethode vor. Wir nutzen die Eigenschaft, dass Kanten im ESDF im Generalized Voronoi Diagram (GVD) Flächen sind und können daraus maximal freie Bereiche extrahieren. Wir fügen dann einen Graph an diese Kanten an und erzeugen somit eine kompakte Darstellung der Umgebung, mit der wir eine Grössenordnung schneller planen können als mit dem schnellsten sampling-basierten Ansatz. Darüber hinaus funktioniert unser Ansatz mit engen Passagen, mit welchen sampling-basierte Methoden in der Regel Schwierigkeiten haben.

Schliesslich demonstrieren wir ein vollständiges System mit allen Komponenten dieser Arbeit, und evaluieren es in realen S&R und industriellen Inspektions-Szenarien. Wir zeigen, dass unser Ansatz speziell für diese Anwendungen geeignet ist und gängige Ansätze übertrifft.

# Acknowledgements

This thesis was by no means done alone, and I would like to thank all those that helped and supported me through this time.

First, a huge thanks to Professor Roland Siegwart for letting me do the thesis at the Autonomous Systems Lab, and building a lab that has been like a family to me for the past years. A huge thank you also to Professor Shaojie Shen, for agreeing to co-examine the thesis, come all the way to Zürich, and doing so much excellent work in the field. I am eternally thankful to Dr. Juan Nieto, for his unwavering support, help and resources, and keeping me going through thick and thin. This thesis would have never seen completion without you.

I had the distinct pleasure to work directly with a wide variety of people during this thesis. There's a few people that require special mention. I'd like to thank Dr. Michael Burri, for teaching me most of what I know about drones, being honest with me about the quality of my ideas, and many bouldering sessions. Dr. Zachary Taylor, for all the technical contributions, pragmatic advice, and all that fun that was had making fun of Kiwi accents and sheep jokes. Alex Millane, for sitting to close to me for so many years, and sharing my pile of garbage. Also the brilliant ideas. Rik Bähnemann for support on the planning side, doing tasks so terrible no one else wanted to do them, teaching me German, and sharing many a beer, in lab and outside. Marius Fehr for doing so much software engineering with me, sitting through massive pull requests, and putting up with me even when it wasn't so easy. Especially at tough times like deadlines, demos, and competitions, I couldn't have had better company.

I'd also like to thank Marco Tranzatto, Christian Lanegger, Karen Bodie, Michael Pantic, Mina Kamel, Dr. Renaud Dubé, Dr. Abel Gawel, Dr. Dominik Honegger, Dr. Gregory Hitz, Dr. Enric Galceran, and Dr. Markus Achtelik for being excellent collaborators and for all the help, advice, and fun times, good times, bad times, terrible integration weeks, demos, robotics competitions, and beers. I'd also sincerely like to thank all of my students, for working so hard, being motivated, and teaching me so much – both technical and personal.

Without a doubt, three more people were instrumental to the thesis: Luciana Borsatti, Cornelia de la Casa, and Michael Riner, without whom, the lab would not run, my problems wouldn't get solved, and the entire LEE J floor would descend into chaos. Additionally, I am very grateful to Dr. Mark Hopflinger for funding the last years of my PhD through armasuisse, and being the best project leader I could imagine.

Finally, I could not have finished without my friends and family from outside the lab. For my dear friends Sybil Ehrensberger, Anya Niafiodava, Dr. Federico

Camposeco, Dr. Andrea Cohen, Nick Spooner, Ashish Anand, and Seraina Steiner, for the fun hikes, spa days, adventures, barbeques, or just hanging out, and for always putting up with my hectic schedule. A massive thanks to Pedro Mendez Montejano, for being a wonderful and patient roommate, the best anyone could ever ask for. To Edo Jelavic, for being an excellent partner-in-crime throughout the whole PhD, but especially for your love and support at the end, and when things got toughest. I wouldn't trade the hiking trips, ski tours, diving, or just sharing fun times with you for the world. And last but certainly not least, to my family: Svetlana Oleynikova, Mikhail Popov, and Dmitriy Ivanov, for supporting this journey even though it has meant I'm very far from home.

Winter 2019 *Helen Oleynikova*

## Financial Support

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7) under grant-agreement n. 608849 (EuRoC). We also received funding from the Swiss Commission for Technology and Innovation (CTI), project n. 16432.1 and the National Center of Competence in Research (NCCR) Robotics, Swiss State Secretariat for Education, Research and Innovation (SERI) n.15.0029. Finally, the author was primarily sponsored by armasuisse S+T research program 56, Unmanned Mobile Systems.

# Contents

# Preface

This doctoral thesis is organized as a cumulative thesis, bringing together multiple previously-published works, which are attached at the end of the report.

The core purpose and ideas of the thesis, including a thorough discussion of the motivation, are presented in Chapter 1. Each contributing paper is discussed in detail, including its context, contributions, and interrelations to other work in Chapter 2. Finally, we summarize the achievements of the papers presented in this thesis and suggest avenues for future work and future research directions in Chapter 3.

Chapter 1

# Introduction

Micro-Aerial Vehicles (MAVs) have enjoyed great attention in the research community, and in recent years, stunning success as commercial products for consumer markets. Their dynamics make them widely applicable in a range of scenarios: fast dynamics and high agility allow them to move quickly and freely in 3D space, easily reaching places difficult for people to inspect. They pose an interesting problem for research because their advantages make them difficult to control: the systems are inherently unstable, and require fast, active stabilization to stay airborne. Having a flying system also limits the potential payload significantly, giving us a system that requires very high-speed processing but can only lift very limited computing payloads. This creates a very rich and interesting area of research, which this thesis aims to contribute to.

Consumer applications of drones have mostly focused on "selfie-drones": aerial vehicles that are able to track and follow a target, with a high-resolution camera on a gimbal for recording video footage, especially of sports. At the beginning of this thesis, commercial platforms were only capable of GPS-based navigation, in open space with no collision avoidance. While this has changed over the course of this PhD, with the DJI Mavic[1] and Skydio R1[2] both able to navigate using visual-inertial odometry in GPS-denied environments, and perform reactive collision avoidance in uncluttered environments, there are still many use-cases not covered by commercial systems, such as close-to-structure inspection and higher-level autonomous operation.

Rather than focusing on the consumer markets as described above, we want to create a navigation framework that can be useful for two core areas of application: search and rescue (S&R) and industrial inspection. Search and rescue is a particularly difficult environment, where no *a priori* knowledge of what the environment could look like is available and it is dangerous for humans to enter. For industrial

[1] www.dji.com/mavic
[2] www.skydio.com

(a) (b) (c) (d)

**Figure 1.1:** Examples of various scenarios this thesis focuses on: (a) and (b) are at a simulated firefighters training area at Wangen an der Aare in Switzerland, (c) is in a forest in Zürich, Switzerland, and (d) is in the Machine Hall of ETH Zürich.

inspection, more prior information may be available, but inspections should be carried out over long periods of time, when the interiors could change. Both applications require operating in cluttered, at least partially unknown environments, and very close to structure.

We aim to address these needs by designing mapping and planning solutions that work together to allow fast, on-board, safe navigation system, in cluttered, GPS-denied environments. We build on existing solutions for trajectory tracking control by using non-linear MPC [42, 43] and local state estimation by using visual-inertial odometry [7, 56]. Instead, we focus on dense 3D mapping and local planning for collision avoidance through these maps.

Much of the focus of this thesis is exploring the interrelation between mapping and planning: how a map representation can be suited to the needs of planning, and how planners can exploit all the information available in the map for better planning performance. We pay special attention to *safety* and planning conservatively – that is, not making any optimistic assumptions about unknown space. We aim to address the needs of navigation in cluttered environments, with special focus on search and rescue, industrial inspection, and forest flight.

## 1.1 Motivation and Objectives

The main objective of this thesis is to create a full local navigation system that is able to create dynamically-feasible timed trajectories for an MAV based on raw dense sensor data and pose estimation. We approach this by presenting two main components: (i) a safe, local navigation planner that is *conservative* with respect to observed space, and (ii) an environment representation that is both fast enough to run on-board and is designed to be used for planning, explicitly storing known and unknown space. From our target applications, we additionally have the re-

**(a)** **(b)**

**Figure 1.2:** Two of the MAV systems used for this thesis, including various sensor configurations. (a) is an Asctec Firefly, with a dual-core 1.7 GHz i7 CPU and a stereo VI Sensor [78] as the only perception. (b) is a custom-built system based on the DJI FlameWheel F550 frame, with an Intel NUC on-board, and outfitted with a VI Sensor, an Intel D400 RGB-D sensor, a thermal camera, and a higher-resolution downward-facing camera.

quirements of operating close to structure and in *very* cluttered environments.

Fig. 1.1 shows examples of the kind of environments we want to be able to safely navigate in. Subfigures 1.1a and b show examples of realistic scenarios for post-earthquake search and rescue, c shows an outdoor GPS-denied forest scenario, and d shows an industrial inspection scene. All of these require safe navigation in complex, cluttered environments.

Furthermore, we have the following specific requirements from the systems side – as our platforms, shown in Fig. 1.2, have low payload, vision-based sensing, and limited computational ability.

**GPS-Denied Navigation** Since our required scenarios are a mix of indoor and outdoor scenes, and some outdoor GPS-denied environments like the forest in Fig. 1.1c, we cannot rely on external global positioning. Instead, we can only estimate the system's pose locally from on-board sensors, which is prone to slow drift over time. The proposed solution must be able to remain safe in the presence of drift.

**Vision-Based Sensing** Due to the low payload of MAVs, they are most often outfitted with cameras as the main sensing modality, as they are a rich sensors with very low power and payload requirements. While cameras work incredibly well for fast state estimation, it is more difficult to extract dense data necessary for planning. Both the stereo vision setup and RGB-D sensors shown in Fig. 1.2 suffer from low field of view and limited sensing range. This means our algorithms have to account for most of the scene being out of the field of view of the platform.

**Conservative Planning** Especially due to the low field of view of the vision sensor, we cannot trust that unobserved areas of the environment are free. This becomes an increasingly unsafe assumption as the environment becomes more cluttered. Therefore, we can only safely traverse space that we have observed as free.

**High Rate Replanning** As we assume no *a priori* knowledge of the environment and treat unknown space conservatively, to make any progress toward a goal, the system must replan often. We aim for a replan rate of approximately 4 Hz, which given a 10 meter sensing range for our sensor, allows us to safely move at a maximum safe speed of 10 m/s.

**Limited Computation** To be useful for Search and Rescue operations, the MAV must be able to behave intelligently even with communication loss or very limited bandwidth. Therefore, all processing must be done on-board. Our systems have Intel x86 CPUs that are comparable to a standard laptop and have no embedded graphics, so all processing should be CPU-only.

**Assistance to Human Operators** Finally, since we do not aim to solve all problems related to robot autonomy, the system should aim to *complement* and *assist* human operators rather than replace them. This entails having map representations that are easy for humans to parse, easy ways to input complex commands, and leaving the high-level decision making to people.

## 1.2 Approach

Our approach presents contributions in two main areas: **planning** that exploits map information, and **mapping** that is specifically designed for planning applications.

For planning, we aim to solve two sub-problems: first, local collision avoidance, and later, global planning. Our approach for local collision avoidance centers on using *local trajectory optimization* on polynomial splines [81], called *loco*. We begin with an initial straight-line guess and then iteratively deform the trajectory to push it out of collision, similar to Ratliff *et al.* [93], but taken to the continuous-time domain with different dynamic constraints. This approach is fast and yields smooth, dynamically-feasible trajectories. The downside is that this local optimization requires (or at least benefits from) obstacle distance and gradient information. While the required distance fields are possible to compute from existing occupancy grid methods such as Octomap [38], even incremental solutions require having a fixed-size upper bound on the map size and computing the distances to the edge of the map [51].

To allow easy planning, we design a map representation explicitly for storing and maintaining distance fields. Our representation, named *voxblox*, has fast access times, can grow dynamically in size, and can handle many different types of data stored in a voxel grid [83]. Based on KinectFusion [74], we use Truncated Signed

Distance Fields (TSDFs) to incorporate distance measurements. We then maintain full Euclidean distances to obstacles using incremental Euclidean Signed Distance Field (ESDF) building based on the incremental occupancy approach of Lau *et al.* [51]. Constructing the ESDFs out of TSDFs rather than occupancy information increases our accuracy substantially, and allows us to model detail in objects finer than the voxel resolution.

We integrate our local planning and mapping approach together to create a system specifically suited for safely navigating in initially unknown environments, which can incrementally and quickly update a dynamically-growing map, and plan paths through observed free space. Since we treat unknown space very conservatively (only areas that we have explicitly observed as free are considered traversable), the local optimization is prone to falling in local minima and being unable to find a path that makes progress toward the goal. This problem is much worse in cluttered environments, where many obstacle occlusions make the visible traversable space very small. To overcome this, we propose to employ a local exploration strategy when the normal path-planning method fails [85]. We design an approach that combines both goal-seeking and maximizing exploration gain to allow us to first explore areas that are likely to lead to the goal. This greatly increases the overall success rate of our planning, especially in long scenarios. Additionally, we show that this outperforms the traditional method of using a conservative local planner with an optimistic (treating unknown space as free) global planner.

Finally, we examine methods to exploit the information stored in an ESDF map. We take the case of global planning, for instance for repeat inspections of the same industrial environment or multiple missions in the same search and rescue scene, where a global map is available. We exploit the property that the ridges of the ESDF form the faces of the generalized voronoi diagram (GVD) [105], which contains the points that are maximally removed from obstacles. We then perform filtering and thinning operations, which allow us to create consistent diagrams even with changing voxel sizes and different noise densities, and fit a sparse topological graph to this diagram. This allows us to perform global planning much faster than possible with sampling-based methods, while maintaining a deterministic planning graph that compactly represents the free space in the environment [86].

This complete system, comprising local planning, global planning, and mapping is summarized in [84] and shown in Fig. 1.3 There has been substantial work in integrating this into a fully-running platform, running everything online and on-board, and some special considerations in terms of control structure, state estimation, and localization.

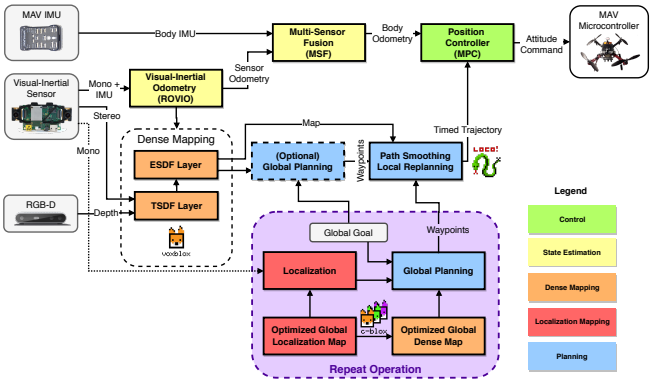



**Figure 1.3:** System architecture, showing the data flow over multiple missions: in white the general or single-mission case, and in purple, the repeated mission case where a globally-optimized prior dense map is available.

Chapter 2

# Contribution

This chapter will detail the contributions of each of the papers presented as part of this cumulative thesis. We will describe the context of the work at the time of publication, novel contributions, and finally how the paper relates to the rest of the thesis and how it has impacted the field.

We present the papers in chronological order, as the development of the algorithms went organically between developing planners that exploit map information, then improving map representations to inform planning algorithms, and continuing this cycle.

## 2.1 Paper I

Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran, "Continuous-Time Trajectory Optimization for Online UAV Replanning". In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

### Context

This paper attempted to address a large gap in the MAV planning literature: fast, safe, dynamics-aware online replanning. While many planning methods were available, most sophisticated ones focused on global planning and were too slow to be used online in active replanning. One popular example is RRT*-based methods for initial waypoint selection [44], and polynomial smoothing through these waypoints to create a dynamically-feasible, smooth path [95]. While this method works well even for planning on long, complex horizons, it takes several seconds to create a complete path. A method addressing the same problem was published at a similar time, using free-space corridor generation through an octomap to rapidly generate trajectories [14]. However, their approach is *optimistic* rather than *conservative*:

unknown space is assumed to be free, and the replanning rate should be fast enough to avoid collisions if previously unseen space happens to be occupied. However, this is inherently unsafe, especially as environments get more cluttered. We aimed to design an algorithm that is conservative, and will plan to stop in known free space in order to always have the executed path be known collision-free.

### Contribution

Our solution focuses on using local trajectory optimization with a polynomial spline representation. We perform local optimization that has two competing costs: a "smoothness" cost, where we aim to minimize some derivative of position of the entire trajectory [66], and a collision cost, where the trajectory should attempt not to collide with obstacles. It is an extension of CHOMP [93], which changes the underlying trajectory representation to a continuous-time one (rather than discrete waypoints), and converts it to a simple unconstrained optimization, as opposed to augmented gradient descent in the original work. We also formulate the collision cost as a line integral over the entire trajectory, allowing us to exploit the smoothness of both the trajectory and the distance field representation. We show very fast (40 ms) total reaction times, including incorporating data into a map, updating the ESDF incrementally, and running the trajectory optimization, which is much faster than previous work.

### Interrelations & Dissemination

This paper is the building block of all subsequent work in this thesis. Paper III [83] later extends this work by changing the underlying map representation to an ESDF that can grow dynamically (in this work, there was a fixed maximum size from the beginning). The original motivation behind creating a new map representation was to allow the ESDF necessary for this optimization method to grow dynamically with no fixed maximum size. Paper IV [85] extended this by introducing an intermediate goal-finding strategy, which attempts to overcome the high failure rate of the local optimization (due to falling into infeasible local minima).

This paper has also had long-lasting impact on the community. The code is available online and open-source[1], including the forest flight sample simulation maps[2]. Others in the community have built on our approach in various ways. Usenko *et al.* have extended the approach to use B-splines rather than Hermetian splines, using our evaluation framework and a custom circular buffer map representation [109], showing a significant speed-up in optimization time. Lin *et al.* follows our general approach to locally optimize collision-free trajectories in a two-stage optimization, but improves success rate substantially by initializing with a free path found by A* [57]. Finally, Morell *et al.* propose ASTRO, which is similar approach using ESDFs to directly optimize collision costs locally on a polynomial spline [71], but

[1] github.com/ethz-asl/mav_voxblox_planning
[2] github.com/ethz-asl/forest_gen

uses constructed free-space corridors from the ESDF rather than planning directly in the map.

## 2.2 Paper II

Helen Oleynikova, Alex Millane, Zachary Taylor, Enric Galceran, Juan Nieto and Roland Siegwart, "Signed Distance Fields: A Natural Representation for Both Mapping and Planning". In *RSS Workshop on Geometry and Beyond*, 2016.

### Context

This paper suggests a better map representation for planning, based on the needs in Paper I [81]. It aims to bring together ideas from both robot planning literature, which in 3D most often uses Octomap [38], and 3D reconstruction and graphics, which in recent years have been taken over by Truncated Signed Distance Field (TSDF) approaches like KinectFusion [74].

The idea behind this work is to suggest that perhaps probability-based voxel grids are not the best representation for planning, and that signed distance fields contain more useful information, such as storing both uncertainty (encoded as the weight) and distance to obstacles. TSDFs hold a voxel grid that, for each voxel, stores both a weight and a distance. They are created by ray-casting pointcloud data into a grid, and assigning positive distance values to voxels in front of the surface, and negative distances behind. These distances are calculated in the *projective* space – meaning the distance along the ray from the sensor center to the surface. As such, they always *overstimate* the true Euclidean distance. Therefore, to minimize error, the TSDF holds distances only up to a small *truncation distance* around the surface.

Euclidean Signed Distance Fields (ESDFs), on the other hand, store true Euclidean distance to obstacles. Our paper explores initial ideas on how to create full ESDFs from TSDFs.

### Contribution

This is a workshop paper about a new way of representing maps, and as such, the main contributions are the ideas to combine building TSDF and ESDFs in the same map. Since both TSDFs and ESDFs are distance fields that encode distance to obstacles, the most naive idea is to simply *not* truncate the TSDF.

We show results of removing this truncation and then apply multiple strategies for fusing scans together, on sample simulated scenarios in 2D. We show that if we simply take the mean of all distance values (as is normally done in the TSDF fusion), we get large systematic errors and an unusable distance field. This is due to the property of projective distances strictly overestimating Euclidean distances, and that this problem becomes much worse the farther the voxel is from the surface boundary. We also attempt another fusion method which simply takes the minimum of all observed distances, which substantially improves the usability

of the map but still leaves many discontinuities in the field and is much more susceptible to noise.

Finally, the strategy that we propose at the end, which has the lowest error, is to create a hybrid E/TSDF, which uses TSDF values near surface boundaries and computes the ESDF values using normal wavefront propagation techniques further from the surface.

#### Interrelations & Dissemination

This workshop paper presented the initial ideas and analysis that led to Paper III [83], which suggests a full method to compute this hybrid E/TSDF, presents error statistics and theoretical bounds on error, and shows a full system able to use this map in 3D, online, and on-board an MAV.

## 2.3 Paper III

Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning". In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

#### Context

This paper aimed to combine the best of both worlds for 3D reconstruction and planning maps for 3D map representations. We wanted to create a system that leveraged recent advances in 3D mapping for high-resolution reconstruction, namely KinectFusion [74] and other TSDF-based approaches, with the information necessary or helpful for planning, which is free-space information and distance information to obstacles in free-space. Traditionally, Octomap [38] has been used for the vast majority of 3D planning applications, but has many shortcomings, most notably look-up speed and aliasing effects. We aim to create a map representation that is both hand-tailored for planning applications, and has the advantages of TSDFs: namely, being easy to visualize and parse for a non-expert, and storing information to a resolution higher than the voxel size.

#### Contribution

We propose three core ideas to fulfill our goals of a flexible SDF-based map representation suitable for planning, which we name *voxblox*:
1. a generic dense voxel representation that supports multiple "layers", storing distance, occupancy, or semantic information and using voxel hashing [76] as the basic representation,
2. a pointcloud integration algorithm that is better suited for larger voxels, speeding up integration times by up to 20x over naive raycasting,

3. and an integration scheme for incrementally generating ESDFs out of TSDFs, adapted from the incremental method proposed by Lau *et al.* [51], and showing higher accuracy than methods based on occupancy information.

This paper takes the ideas from Paper II [82] and creates a full usable 3D system, shown running online an on-board an MAV with real sensor data, and then used for planning using the method presented in Paper I [81]. Additionally, for the applications to robot path-planning, we offer an analysis of the impact of all the simplifying assumptions we use in computing the ESDF, and suggestions on error bounds to apply in selecting safety margins.

#### Interrelations & Dissemination

Similar to how Paper I [81] was the basis of all the planning methods in this thesis, this is the core map representation we use for our remaining work. Specifically, Paper IV [85] uses voxblox as the core map representation, exploiting information in both the TSDF and ESDF to track unknown space, and Paper V [86] exploits the properties of the ESDF to generate Generalized Voronoi Diagrams from maps, and from there create sparse graphs that can be used for fast global planning.

This work was made available open-source since publication[3], and has since enjoyed a large amount of improvements, new features, and support from the author and collaborators, most notably including faster integration methods and an implementation of pointcloud-to-SDF ICP for fine pose refinement.

Others have also built on voxblox as a mapping framework. Blochliger *et al.* uses voxblox for topological mapping based on previous trajectories through the environment [5]. Millane *et al.* extends voxblox to use sub-maps for consistent dense mapping in the presence of loopclosures [68]. Furrer *et al.* uses voxblox as a dense representation for object models, and uses the properties of signed distance fields to fuse and verify correct alignment of 3D models [29].

Finally, Florence *et al.* has evaluated their proposed sparse map representation against our mapping framework as a standard solution [23].

## 2.4 Paper IV

Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto, "Safe Local Exploration for Replanning in Cluttered Unknown Environments for Micro-Aerial Vehicles". In *IEEE Robotics and Automation Letters (RA-L)*, 2018.

#### Context

This paper aims to overcome the inherent downsides of conservative, local trajectory optimization methods for collision avoidance: the high failure rate from falling into local minima. This is a particular problem in very cluttered environments with a narrow field of view: often, there does not exist a visible path

[3] github.com/ethz-asl/voxblox

towards the goal that passes through only known space. We explore different intermediate goal-finding strategies when naive local pathfinding fails, in order to increase overall success rate to get to the goal. One key contribution of this work is that it deals with significantly denser/more cluttered environments than other approaches. Where other methods that evaluate on randomized maps end their experiments at 10% obstacle density [81], [14], that is where our evaluations *begin*, and we do experiments on up to 50% obstacle density. Coupled with a narrow field of view from our vision-based sensor, we attempt to tackle a very difficult problem.

#### Contribution

The approach we take to overcome issues with local navigation in cluttered, mostly unseen environments is to borrow concepts from exploration literature. We analyze multiple potential intermediate goal finding strategies, which select shorter-term goals when no path toward the global goal can be found, including random, RRT [44], and next-best-view planner (NBVP) [4]. Our proposed method is a simplified version of the evaluation exploration that NBVP proposes: We sample multiple near-by points and choose the one that maximizes both potential exploration gain and progress towards the global goal. We found that our strategy was faster to evaluate and increased the success rate of the overall planning problem by up to 70% over not selecting intermediate goals. The complete replanning system is evaluated on a real platform in a variety of cluttered environments, including an office space and multiple forest scenes.

#### Interrelations & Dissemination

This paper combines Paper I [81] and Paper III [83], fully taking advantage of the information available in the voxblox TSDF and ESDF maps to improve local planning. Mostly it aims to overcome a crucial shortcoming in the local continuous optimization approach, where it frequently fails in very cluttered environments when no path through the known space toward the goal is seen. The main finding is that optimistic global planners do not work very well when the environment gets more cluttered, and more intelligent strategies are necessary in such cases. The approach and evaluation benchmarks are available open-source[4].

## 2.5 Paper V

Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto, "Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning". In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[4] github.com/ethz-asl/mav_voxblox_planning

### Context

Rather than focusing on local planning and collision avoidance like other papers in this thesis, here we focus on global planning through a previously-known map. We exploit the properties of ESDFs to create deterministic, 3D sparse topology graphs of the free-space to significantly speed up planning. There has been extensive literature on creating sparse topologies of 2D spaces before [108], [52], [41], [58]. However, literature in 3D has been comparatively limited, and either depends on the trajectory used to generate the map [5] or has restrictions on maximum clearance between obstacles [15]. We aim to extend similar methods to what has been done in 2D, exploiting only the information in the map, and design a method that is less sensitive to noise and differences in voxel sizes, but in full 3D.

### Contribution

The work proposes a novel method to extract generalized voronoi diagrams, voxel skeletons, and sparse topology graphs from ESDF maps (voxblox, in this case [83]). We exploit the property of ESDFs that their ridges form the faces of a Generalized Voronoi Diagram (GVD). While in 2D, a GVD creates lines that are equidistant from obstacles, in 3D, it creates surfaces. Since we are interested in getting the sparsest representation we can, we focus on how to extract sparse 3D graphs on the edges of the GVD surface.

We use tools from the skeletonization literature [105] to deal with one of the largest problems in GVD-based skeleton extraction: sensitivity to noise. Since we deal with real noisy sensor data rather than perfect CAD models, this problem is of significant interest.

We propose a filtering approach, new diagram removal templates, and other techniques for counteracting both noise and resolution changes. We then build a sparse graph, consisting of vertices and edges, from the thinned skeleton diagram which we can use for planning. The size of this diagram remains relatively stable regardless of underlying voxel size or noise level. Finally, we demonstrate how this sparse graph can be combined with polynomial smoothing (based on [81], [85]) to quickly create dynamically feasible and collision-free trajectories through the environment. The initial search through the sparse graph is orders of magnitude faster than sampling-based methods such as RRT [44].

### Interrelations & Dissemination

This work moves away from doing only local replanning to solving the global planning problem as well. The approach is significantly faster than RRT, and when combined with path smoothing methods, leads to long, dynamically-feasible trajectories that can be planned very quickly. Combined with the other papers in this thesis, this gives us a complete mapping and planning system suitable for repeated inspections and missions in the same environment, while still being able to cope

with local changes. The work is made available open-source[5]

## 2.6 Paper VI

Helen Oleynikova, Zachary Taylor, Alexander Millane, Roland Siegwart, and Juan Nieto, "A Complete System for Vision-Based Micro-Aerial Vehicle Mapping, Planning, and Flight in Cluttered Environments". In *arXiv:1812.03892*, 2018.

### Context

The final paper in the collection is an unpublished technical report, summarizing the complete system described in this thesis, including considerations from other aspects such as control and state estimation.

It is similar in structure and purpose to two recent system papers from other groups. The first, Lin *et al.* [57] presented a complete system, with special focus on the state estimation, but uses a more standard occupancy grid approach for mapping. Our work instead focuses on the interrelation of mapping and planning (also with other components), chiefly addressing issues with narrow field of view sensors in very cluttered environments.

More recently, Mohta *et al.* [69] published their system for the DARPA Fast Flightweight Autonomy (FLA) competition. Their focus is also on GPS-denied navigation, but with a focus on flying as fast as possible. A core difference is the sensor suite: they use a 360° field of view LIDAR, which removes many of the challenges with narrow field of view sensing that we attempt to tackle in our work. Their mapping approach also differs substantially: they keep only a local map, and attempt to decompose free-space in this map into overlapping convex regions, which works very well in sparser environments but does not scale to very cluttered areas. Their global map is also only kept in 2D in order to be able to push the robot out of dead ends, while we keep a complete global map to be able to perform full 3D global planning.

Ultimately, the most suitable system depends on sensor suite and application.

### Contribution

The core contribution of this work is to present the complete system, from hardware and sensors, to state estimation and control, to local and global mapping and planning. The aim to serve as a reference for others attempting to replicate the system and ensure that all software is available open-source.

In addition to describing the complete system, we extend our previous work in sparse skeleton topology to significantly speed up sparse graph generation and improve quality of the graph, and have a more thorough discussion of how this graph can be used for planning. We additionally extend our local collision avoidance method, Loco, to also function as a path smoothing method – allowing it to

[5] github.com/ethz-asl/mav_voxblox_planning

take a list of waypoints from a global planner and output a dynamically-feasible, collision-free path.

Finally, we focus on benchmarking various global planning methods and path smoothing methods together. The datasets we test on are optimized global maps from three real search and rescue and industrial inspection scenarios: two at the military rescue training ground at Wangen an der Aare in Canton of Bern, Switzerland, and one from the ETH Machine Hall in Zürich. We show that our proposed topological skeleton-based global planning and local continuous path smoothing have computation time and success rate advantages over competing methods.

#### Interrelations & Dissemination

This paper attempts to bring together all of the previous work into one complete system, and specifically extend the local planning method in Paper I [81] to be suitable for path smoothing, provide extra considerations about how to treat unknown space in ESDFs from Paper III [83], and siginfiicantly speed up and improve the graph construction method presented in Paper V [86]. Though it is currently only available on arXiv, we intend to extend this work with more results on local planning in completely unknown scenes and additional real-world experiments, and then submit to a journal.

## 2.7 List of Publications

There are multiple publications that have resulted directly from the thesis, or received significant assistance from the author. They are sorted by first author and by year.

### 2.7.1 Publications Included in this Thesis

- H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran. Continuous-time trajectory optimization for online uav replanning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016
- H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart. Signed distance fields: A natural representation for both mapping and planning. In *RSS Workshop on Geometry and Beyond*, 2016
- H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017
- H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto. Safe local exploration for replanning in cluttered unknown environments for micro-aerial vehicles. *IEEE Robotics and Automation Letters*, 2018

- H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto. Sparse 3d topological graphs for micro-aerial vehicle planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018

- H. Oleynikova, Z. Taylor, A. Millane, R. Siegwart, and J. Nieto. A complete system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments. *arXiv preprint arXiv:1812.03892*, 2018

### 2.7.2 Other Publications

- H. Oleynikova, M. Burri, S. Lynen, and R. Siegwart. Real-time visual-inertial localization for aerial and ground robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept 2015

- M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015

- M. Burri, J. Nikolic, H. Oleynikova, M. W. Achtelik, and R. Siegwart. Maximum likelihood parameter identification for mavs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4297–4303. IEEE, 2016

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

- A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots*, pages 1–16, 2016

- A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena. C-blox: A scalable and consistent tsdf-based dense mapping approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018

- C. Witting, M. Fehr, R. Bähnemann, H. Oleynikova, and R. Siegwart. History-aware autonomous exploration in confined environments using mavs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018

- F. Ruetz, E. Hernández, M. Pfeiffer, H. Oleynikova, M. Cox, T. Lowe, and P. Borges. Ovpc mesh: 3d free-space representation for local ground vehicle navigation. *arXiv preprint arXiv:1811.10266*, 2018

- C. Papachristos, M. Kamel, M. Popović, S. Khattak, A. Bircher, H. Oleynikova, T. Dang, F. Mascarich, K. Alexis, and R. Siegwart. Autonomous exploration and inspection path planning for aerial robots using the robot operating system. In *Robot Operating System (ROS)*, pages 67–111. Springer, 2019

## 2.8 List of Supervised Students

The author also supervised a number of students during the doctoral studies, which constituted a significant portion of the author's time. We give a list of the students and their project names below, and give citations to those works which have resulted in a publication.

### Master Thesis

*Master student, 6 months full time*

- Sebastian Inderst (Fall 2017): "Online Planning for UAV in Unknown Cluttered Environments"
- Fabio Ruetz (Spring 2018): "Local Navigation on Pointclouds" [97]
- Nils Funk (Spring 2018): "Safe Motion Planning in Real-time Mapped Euclidean Signed-distance Fields"
- Jaeyoung Lim (Fall 2018): "High-Speed Autonomous Navigation in Unknown Environments"
- Victor Reijgwart (Fall 2018): "Real-time, Consistent, Volumetric SLAM Onboard MAVs for the MBZIRC 2020 Challenge"

### Semester Thesis

*Master student, 3-4 months part time*

- Dimitris Gryparis (Fall 2015): "High Level Control of an MAV through Marker-Based Visual Commands"
- Marco von Atzigen (Fall 2015): "Rapid Replanning for Unknown Obstacles Onboard MAVs"
- Antoine Seewer (Spring 2016): "Motion Planning for UAV landing on a Moving Platform"
- Michael Pantic (Spring 2016): "Detection of Moving Targets from UAV and Tracking using Priors"

- Sebastian Inderst (Spring 2017): "UAV Online Trajectory Optimization for Forest Flight"
- Nicolas Scheidt (Fall 2017): "Coverage Planning for 3D Terrain Reconstruction with Micro Aerial Vehicles"
- Christopher Bilberg (Fall 2017): "Onboard Mapping and Localization for MAVs Using Maplab"
- Christian Witting (Fall 2017): "Viewpoint Aware Exploration" [114]
- Leonie Traffelet (Fall 2018): "Multi-Camera VIO with Rovio"
- Nikhilesh Alatur (Fall 2018): "Autonomous Aerial Firefighting with a UAV"

## Bachelor Thesis

*Bachelor student, 3-4 months part time*

- Tobias Spiegelburg (Fall 2015): "Obstacle Avoidance on MAVs using Repulsive Force Approach"
- Alessandro Morra (Spring 2018): "Autonomous Single Floor Exploration by a Two-Wheeled Robot with a Local and Global Planner Approach"

## CSE Seminar in Robotics

*Master student, literature review, 3-4 months part time*

- Andreas Hug (Spring 2017): "Trajectory Planning for Quadrotors"
- Simon Frasch (Spring 2017): "Multi-Robot SLAM"

Chapter 3

# Conclusion and Outlook

This thesis provides a complete system, including mapping and planning, for local collision avoidance in cluttered, unknown environments on-board MAVs. We focused on light-weight solutions that are able to run in real-time onboard, trading off global optimality for execution speed. The core contributions of this thesis can be split into 3 categories:

**Local Continuous Trajectory Optimization** We proposed a method that uses gradient-based local trajectory optimization (named *loco*) to create smooth, collision-free, dynamically-feasible polynomial splines through the environment. A key distinction between our method and other approaches in literature is that we are completely conservative: all trajectories end in known free space. We also investigated ways to improve the success rate of this method in very cluttered environments, as many occlusions make it unlikely to find a path toward the goal. We proposed a local exploration strategy that can be employed when the underlying optimization fails, greatly increasing our success rate in reaching the goal in complex environments.

**Signed Distance Field Mapping** Incorporating the requirements of our planning strategy, we designed a flexible map representation, called *voxblox*, built on signed distance fields. New data is incorporated into a TSDF layer, and we propose fast ways to handle large voxel grids, which are usually employed for planning applications but not for 3D reconstruction. We then describe a method for generating accurate and fast incremental ESDFs from this TSDF layer, and show that we have lower errors than doing the same from occupancy maps. This flexible mapping framework has been used for a variety of applications, such as MAV planning, object reconstruction, and ground robot navigation.

**Global Topology Extraction** Finally, we propose a way to exploit our map repre-

sentation to speed up global planning by orders of magnitude for previously-known maps. We propose using the property of GVDs being the ridges in the ESDF to compute a simplified, sparse topological graph of the traversable free-space in the map. This sparse graph can then be used for very fast global planning, and then combined with *loco* for generating smooth feasible trajectories for the MAV.

## 3.1 Future Work

There are many avenues for future work, categorized into two sections below: planning and mapping.

### 3.1.1 Planning

**Guarantees on Non-Collision** In the current system, assuming a static environment, the MAV will never collide because trajectories are collision-checked again before being sent to the robot. If a collision is detected, the old plan (which always stops in known free space) is executed. However, this constraint could be put into the optimization problem as a hard constraint rather than a soft constraint that is later checked for violation.

**Convex Free-Space Estimates** The topological graph from Paper V creates a sparse graph through the traversable free-space, but could be further extended to create convex free-space estimates of the entire traversable corridor by associating distances with edges and vertices in the graph. This could make it possible to better distinguish between two possible paths through the graph, for instance to maximize clearance rather than minimize minimum distance.

**Dynamic Obstacle Modeling and Avoidance** Our approaches assume either a static or a slowly dynamic environment, meaning that the scene changes slowly enough that we are able to replan around it. However, operating in human environments such as crowds would require explicit modeling of fast dynamic objects such as people. We could extend our methods to detect and predict dynamic motion, and incorporate these motion predictions into our planner.

**Planning in Contact with a Manifold** The TSDF representation implicitly assumes that the world consists of continuous surfaces. We can use this for aerial manipulation applications, where many types of sensing require being in contact with a surface to get measurement data, such as assessing material properties for bridge inspection. We could use our maps to ensure that our plans always stay in contact with the surface manifold.

**Uncertainty-Aware Planning** One natural extension is to consider various types of uncertainties in the planning problem. There are many sources of uncertainty

in the system: for instance, state estimation uncertainty, inaccuracies in control, and noise in the map. We could add terms to the optimization problem to not only minimize collision cost, but also consider how any of the three uncertanties mentioned would evolve in different parts of the environment.

**Active Perception** Taking this one step further could mean planning not just for collision avoidance, but to maximize some information gain. Examples could include planning to minimize state uncertainty (by observing as many as possible feature-rich areas), to maximize reconstruction quality, or to facilitate object detections.

**Intelligent Exploration** In Paper IV, we employ a standard sampling-based exploration strategy: sample many free-space points in the environment, and select that which gives the largest exploration gain. We coupled it with goal-seeking behavior for our purposes, but there are many possible other ways to formulate the problem which would take advantage of dynamical constraints of the platform (i.e., try to explore the space while minimizing turns), semantics of spaces, or user-defined sub-objectives which would lead to more predictable behavior for the exploration algorithm.

### 3.1.2 Mapping

**Predicting Unknown Space** Our work focuses on being safe and conservative in unknown environments, and as such always plans to stop in known free space. However, if we could use prior knowledge to predict the structure of unknown space, we could plan faster and more dynamic motions into unexplored areas [96].

**Uncertainty Estimates in Mapping** Our mapping framework contains uncertainty estimates in terms of weights, which contain information about how often a point has been observed and how confident we have been in the measurements based on the sensor model. However, what we do not consider is the state estimate uncertainty at the sensor capture time. This could improve reconstruction quality.

**Multi-Resolution Dense Mapping** We use a fixed voxel size, generally from 10 to 20 cm, for most of the planning applications. However, voxblox is designed to allow multiple layers of different voxel resolutions. Having this multi-resolution map would be especially helpful for mobile manipulation applications, where most of the environment only needs to be modelled coarsely, but surfaces on which manipulation would take place would need a much higher-resolution model.

**Consistent Sparse and Dense Mapping** One issue this thesis did not address is how to bridge the gap between drifting local estimates and global localizations. One key problem is to keep the dense map consistent with a changing

pose graph in the presence of loop closures. Millane *et al.* [68] made the first steps to addressing this problem by creating local submaps and intelligently fusing them based on covariance estimates in the pose graph from ORB-SLAM [73], but there is much work to be done to make this solution general and more widely applicable.

**Localization in Dense Maps** In a similar vein, being able to do localization and loop-closure in dense maps could remove the need for maintaining a sparse landmark map along-side the dense map. There are also environments where dense localization would be very informative, for instance areas of distinct geometric structure but few appearance features.

# Part A

PUBLICATIONS

Paper

# Continuous-Time Trajectory Optimization for Online UAV Replanning

Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran

**Abstract**

Multirotor unmanned aerial vehicles (UAVs) are rapidly gaining popularity for many applications. However, safe operation in partially unknown, unstructured environments remains an open question. In this paper, we present a continuous-time trajectory optimization method for real-time collision avoidance on multirotor UAVs. We then propose a system where this motion planning method is used as a local replanner, that runs at a high rate to continuously recompute safe trajectories as the robot gains information about its environment. We validate our approach by comparing against existing methods and demonstrate the complete system avoiding obstacles on a multirotor UAV platform.

Published in:
*IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016
DOI: 10.1109/IROS.2016.7759784

## 1 Introduction

Multirotor UAVs are gaining wide acceptance not only as research platforms, but also for use in various real-world applications. Despite recent progress in on-board state estimation, planning, and control, many current UAV systems still require either an empty environment or perfect knowledge of one *a priori*. This limits their safety and utility in unstructured, unknown environments.

In this paper, we focus on the problem of planning safe avoidance trajectories for a multirotor helicopter (multicopter) in partially known or unknown environments. For example, use cases such as high-speed forest flight require low-latency motion planning, as these environments are often densely populated and obstacles frequently occlude one another [45].

Motion primitive methods have been a common choice for online replanning on fixed-wing and multirotor platforms, since they can be executed quickly and each "primitive" can be constructed to be dynamically feasible [91] [1]. However, such methods require discretizing the state-space, which requires a huge motion library or having the controller track from nearest start state, which [1] cites as being responsible for up to 20% of the failures of their fixed-wing collision avoidance system. For our approach, we overcome the need for state-space or time discretization by choosing a continuous-time basis function to express our trajectories, and plan from arbitrary points in the state space to allow greater flexibility for online replanning.

We draw inspiration from trajectory optimization methods, such as CHOMP [116], which locally minimize collision and smoothness costs on a discrete-time trajectory. These planners are most commonly used for solving manipulation problems, where most of the constraints are kinematic rather than dynamic. Kinematic constraints can be simply expressed in discrete time, while dynamic constraints are more naturally suited to continuous-time representations (and avoid unnecessary numerical differentiation errors).

For UAV flight, it is advantageous to use continuous-time trajectory representations like polynomials [66]. These basis functions provide continuity up to a high derivative, fast evaluation at any given time, and a very small number of parameters are needed to describe even long and complex trajectories. By varying only one parameter (segment time), it is possible to ensure that these trajectories are dynamically feasible given a simplified model of multicopter dynamics.

We propose a method of continuous-time trajectory optimization that allows the real-time generation of safe avoidance trajectories. Our method uses a two-part objective function, minimizing both a derivative of position and cost of collision with the environment. We also show this method as integrated into a local replanning system, where we use this local trajectory optimization to modify an initial plan in the presence of previously unknown obstacles.

Our approach runs in real-time (under 50 ms for a complete planning cycle), produces continuous-time trajectories, and is able to plan from and to arbitrary states without a need for discretization in the workspace or state-space.

The contributions of this work are as follows:

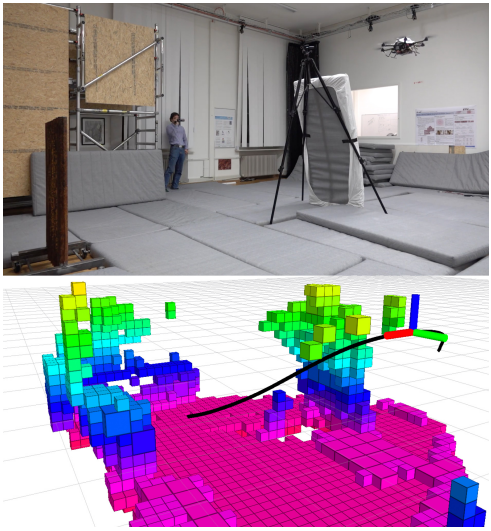


**Figure 4.1:** Results of an experiment showing our online local replanning system. The UAV starts from behind the large mattress, which blocks its view of the iron obstacle (left) behind. The planning goal is set inside the iron obstacle, but as soon as the UAV observes that it is occupied, it replans to stop in front (planned path seen in black). This shows the ability of our system to avoid newly-detected obstacles and find feasible goal positions online.

- A continuous-time polynomial formulation of a local trajectory optimization problem for collision avoidance, capable of running in real-time on a real multicopter.
- A complete system with this as local replanning component, which continuously computes collision-free trajectories around any newly detected obstacles.
- Evaluation against existing trajectory optimization and planning algorithms, and experiments on a real world platform (Fig. 4.1).

## 2 Related Work

3D path planning approaches for UAVs can be broadly classified into several categories, such as sampling-based methods (often followed by smoothing), trajectory optimization methods, and method based on motion primitives. Here we discuss

motion planning methods related to our work, with emphasis in suitability for real-time replanning with a dynamically updating map.

Sampling-based planning followed by a smoothing step which ensures dynamic constraints are met is commonly used for 3D global planning on UAVs [95]. Approaches such as running RRT-based methods to generate a visibility graph, followed by fitting high-order polynomials through the waypoints (graph vertices) are shown to outperform traditional RRT methods in control space in terms of execution time [95]. Recent speed-ups to the polynomial optimization have also allowed such combined planners to run in almost real-time, taking only a few seconds to generate long global plans [8]. While these methods generally produce high-quality plans and are probabilistically complete, they are still too slow for some online applications like real-time avoidance.

Closest to our proposed approach are discrete-time trajectory optimization methods. CHOMP, a trajectory optimization-based motion planner that revived interest in this class of planner in recent years, uses a two-part objective function with a smoothness and collision cost, and performs gradient descent with positions of discrete waypoints as parameters [116]. In order to speed up convergence time to a feasible plan, and ensure smoothness of the final solution, each gradient descent step is multiplied by a Riemennian metric in order to ensure smooth, incremental updates. Also based on trajectory optimization, STOMP is a gradient-free method that samples candidate trajectories and minimizes a cost function by creating linear combinations of the best-scoring candidates [40]. A more recent advent in trajectory optimization for collision-free planning breaks up the workspace into free convex regions and performs Sequential Quadratic Programming (SQP) to converge to a solution faster than the previous two methods [99]. Unfortunately, this requires a pre-built map with pre-computed convex regions, which is difficult to achieve in real-time.

Another approach to finding a low-cost path is to cheaply generate many path candidates, and choose the best of the candidates based on an objective function. This has been done for finding good polynomial trajectories to enable quadrotor ball juggling [72], selecting locally lower-cost trajectories to track a global plan in rough terrain [48], and choosing the safest trajectories for autonomous vehicles in traffic [100]. However, these approaches rely on randomly-sampled trajectories finding collision-free paths, which is an assumption that may not hold in very cluttered environments.

An alternative is to solve the optimal control problem using mixed-integer programming, where the workspace is again broken up into convex regions and a global optimum including some linearized or simplified version of the system dynamics is found [94], [50]. These approaches generally give dynamically-feasible and collision-free trajectories, and it is even possible to make guarantees on their safety [2, 106], however require a map representation which is very costly to compute and generally have long runtimes (on the order of magnitude of minutes).

A class of methods commonly used for replanning are those based on motion primitives. The state-space of the UAV is discretized into a state lattice with motion primitives forming edges in the graph, and standard graph search algorithms

such as A* and AD* are used to find a feasible solution through this graph. This has been shown in multicopters for navigating through partially known environments [64, 91] and on fixed-wing airplanes for navigating through a forest while always safely being able to perform an emergency turn-around maneuver [90]. Another work shows flying high-speed through a forest using only on-board vision and planning, picking collision-free next maneuvers from a motion library [1], where they cite insufficient richness of the motion primitive library and discretization in the start state as responsible for 50% of the experimental failures.

A drawback of these approaches is the need to discretize both the workspace and state space (for example, motion primitives can only be generated for a finite number of start velocities and end velocities), and the performance of the algorithm is tightly linked to how many motion primitives are generated. Although we do use a discretized workspace representation, our approach does not require such discretization, nor does it require discretization in time or state-space, giving the possibility of a wider range of solutions to be found.

# 3 Continuous-Time Trajectory Optimization Algorithm

Our approach focuses on optimizing high-degree polynomial trajectories made out of several segments, as inspired by [95]. The trajectory is essentially a high-order polynomial spline, with $C^D$ continuity, where $D$ is the derivative we attempt to minimize. These high-order splines are generally used for global trajectory generation, and have many advantages including the ability to specify velocities, accelerations, and lower derivatives at waypoints, very fast evaluation times, and compact representation of long and complex trajectories. While a closed-form solution exists to minimizing the sum of squared derivatives of such a spline, we expand the problem to also contain information about the environment to generate a locally optimal safe trajectory.

## 3.1 Problem Formulation

Instead of considering the full dynamics of a multicopter, we follow the work of Mellinger and Kumar [66] to plan in a reduced space of *differentially flat outputs*. This allows us to plan only in $\mathbf{R}^3$ and handle yaw separately.

Therefore, we will consider a polynomial trajectory in $K$ dimensions, with $S$ segments, and each segment of order $N$. Each segment has $K$ dimensions, each of which is described by an $N$th order polynomial:

$$f_k(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \dots a_N t^N \tag{4.1}$$

with the polynomial coefficients:

$$\mathbf{p}_k = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_N \end{bmatrix}^\top . \tag{4.2}$$

Given this trajectory representation, we seek to find the set of coefficients $\mathbf{p}^*$ that minimize an objective function $J$. In our case, similar to CHOMP [116], our

objective function has two components: a part that attempts to minimize a derivative $D$, $J_d$, and a part that attempts to minimize collisions with the environment, $J_c$.

$$\mathbf{p}^* = \underset{\mathbf{p}}{\operatorname{argmin}}\, w_d J_d + w_c J_c \tag{4.3}$$

The following sections will present our choices of objective costs, $J_d$ and $J_c$, optimization method, and map representation to solve this problem in real-time.

## 3.2 Method

As described in [95], we express the polynomial not in terms of its $N+1$ coefficients, but in terms of its end-derivatives to allow us to pose the derivative minimization problem as an unconstrained quadratic program (QP), which is significantly faster to solve than the constrained dual of this problem.

We can map between polynomial coefficients and end-derivatives using the $\mathbf{A}$ matrix, and rearrange the end-derivatives into a free ($\mathbf{d}_P$) and fixed ($\mathbf{d}_F$) blocks using a mapping matrix $\mathbf{M}$:

$$\mathbf{p} = \mathbf{A}^{-1}\mathbf{M}\begin{bmatrix}\mathbf{d}_F \\ \mathbf{d}_P\end{bmatrix}. \tag{4.4}$$

The construction of matrices $\mathbf{A}$, $\mathbf{M}$ (and $\mathbf{R}$ below) is addressed in [95]. The fixed derivatives $\mathbf{d}_F$ are given from the fixed end-constraints, like start and end velocities and accelerations, while the free derivatives $\mathbf{d}_P$ are the parameters we optimize.

In order to incorporate costs from collisions with the environment, we use a minimization problem with two costs:

$$\mathbf{d}_P^* = \underset{\mathbf{d}_P}{\operatorname{argmin}}\; w_d J_d + w_c J_c \tag{4.5}$$

where $J_d$ is the cost due to integrated squared derivative terms (if minimizing snap, integral of squared snap along the trajectory), $J_c$ is the cost due to collisions, and $w_d$ and $w_c$ are the weighing terms for each part of the cost.

The objective $J_d$ can be calculated via the following:

$$\begin{aligned} J_d \quad = \quad & \mathbf{d}_F^\top \mathbf{R}_{FF}\mathbf{d}_F + \mathbf{d}_F^\top \mathbf{R}_{FP}\mathbf{d}_P + \\ & \mathbf{d}_P^\top \mathbf{R}_{PF}\mathbf{d}_F + \mathbf{d}_P^\top \mathbf{R}_{PP}\mathbf{d}_P \end{aligned} \tag{4.6}$$

where $\mathbf{R}$ is the augmented cost matrix, and $\mathbf{R}_{XX}$ denotes the appropriate blocks within this matrix.

The Jacobian of $J_d$ with respect to the parameter vector can be computed as follows:

$$\frac{\partial J_d}{\partial \mathbf{d}_P} = 2\mathbf{d}_F^\top \mathbf{R}_{FP} + 2\mathbf{d}_P^\top \mathbf{R}_{PP}. \tag{4.7}$$

The derivative costs are independent for each axis, and are accumulated over all $K$ dimensions of the problem.

To represent collision costs, we use a line integral of the potential function $c(\mathbf{f}(t))$ over the arc length of the trajectory.

Since our environment is represented as a discrete voxel grid (see Section 3.3), we need to sample the trajectory and test at least one point within each voxel along the trajectory.

To do so, we transform the trajectory from end-derivatives into workspace coordinates. For each axis $k$ at a time $t$:

$$\mathbf{T} = [t^0, t^1, t^2, ..., t^N] \tag{4.8}$$

$$f_k(t) = \mathbf{T}\mathbf{p}_k \tag{4.9}$$

$$\mathbf{f}(t) = \begin{bmatrix} f_x(t) & f_y(t) & \dots \end{bmatrix} \tag{4.10}$$

We also compute the velocity at each time $t$, using a matrix $\mathbf{V}$, which maps a vector of polynomial coefficients of a function to the polynomial coefficients of its derivative.

$$v_k(t) = \dot{f}_k(t) = \mathbf{T}\mathbf{V}\mathbf{p}_k \tag{4.11}$$

$$\mathbf{v}(t) = \begin{bmatrix} v_x(t) & v_y(t) & \dots \end{bmatrix} \tag{4.12}$$

The collision cost is then the line integral below, integrated over each segment $m$ (where $t_m$ is the end time of the segment):

$$\begin{aligned} J_c &= \int_S c(\mathbf{f}(t))ds \\ &= \int_{t=0}^{t_m} c(\mathbf{f}(t)) \left\| \dot{\mathbf{f}}(t) \right\| dt \\ &= \sum_{t=0}^{t_m} c(\mathbf{f}(t)) \left\| \mathbf{v}(t) \right\| \Delta t \end{aligned} \tag{4.13}$$

where $c(\mathbf{f}(t))$ is the potential cost described in [116].

Finally, using the product and chain rules, we obtain the Jacobian for each axis $k$:

$$\begin{aligned} \frac{\partial J_c}{\partial \mathbf{d}_{Pk}} = & \sum_{t=0}^{t_m} \|\mathbf{v}(t)\| \ \nabla_k c \, \mathbf{T}\mathbf{L}_{PP}\Delta t + \\ & c(\mathbf{f}(t)) \frac{v_k(t)}{\|\mathbf{v}(t)\|} \mathbf{T}\mathbf{V}\mathbf{L}_{PP}\Delta t. \end{aligned} \tag{4.14}$$

Here $\mathbf{L} = \mathbf{A}^{-1}\mathbf{M}$, or the complete mapping matrix between end-derivatives and polynomial coefficients. $\mathbf{L}_{PP}$ refers to the block of the right-side columns of the matrix, corresponding to the columns which operate on the free parameters $\mathbf{d}_P$.

We use a heuristic to estimate the segment times, $t_m$, to meet dynamic constraints and we hold these times fixed during the optimization.

## 3.3 Map Representation

Map representation and choice of potential cost function is central to the algorithm described above. Naturally, the potential cost function must be smooth, but its gradient must also be able to push trajectories out of collision. We use the potential described in [116], which is a function of an Euclidean Signed Distance Field (ESDF) value, $d(\mathbf{x})$, at a point in 3D space, $\mathbf{x}$, and $\epsilon$ is a constant value specifying the obstacle clearance past which space is considered free.

$$c(\mathbf{x}) = \begin{cases} -d(\mathbf{x}) + \frac{1}{2}\epsilon & \text{if } d(\mathbf{x}) < 0 \\ \frac{1}{2\epsilon}(d(\mathbf{x}) - \epsilon)^2 & \text{if } 0 \leq d(\mathbf{x}) \leq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

We use a voxel-based map representation, as they can be built and maintained quickly. To ensure that a trajectory does not collide with the environment, we must check each voxel along the trajectory. Note that our continuous-time approach still presents advantages over discrete-time methods, as we are flexible in how often we sample the trajectory for collisions, and can change this interval between iterations. We choose to evaluate the function along every arc length point $\Delta s$ equal to the map voxel resolution. This significantly speeds up computation without compromising safety.

## 3.4 Optimization

In any nontrivial environment, the optimization problem in (4.5) is likely to be non-convex and highly nonlinear. For minimizing the function, we choose to use a quasi-Newton method like BFGS [101] (though other simpler methods like gradient descent can also be used).

However, all solutions found with such methods are inherently *local* solutions – and they are prone of falling into local minima depending on the initialization. Therefore, in order to increase the chances of finding a feasible (non-colliding) solution, we do several random restarts where we perturb the initial state by a random quantity, and then select the lowest-cost trajectory as the final solution. A more thorough discussion on the necessity of random restarts for local trajectory optimization is offered in [99].

# 4 Replanning System

In this section, we introduce the complete system that makes it possible to run our local replanning method online, in real-time on dynamically updating map data. First, we introduce how to build the map and its companion distance field, then we discuss using a global plan as input into the local replanner, and finally, how to select start and end points for replanning.

## 4.1 Incremental Mapping

As mentioned in Section 3.3, we require an Euclidean Signed Distance Field (ESDF) to compute the collision potentials. Our map representation is an Octomap [38], which contains voxels in one of three states: free, unknown, or occupied.

When first constructing the map, we fill in the occupancy of all cells and compute the distances for the complete map, which is computationally expensive. In order to allow our algorithm to run in real-time, we track changed nodes in our Octomap representation and invalidate all voxels in the ESDF that have those nodes as parents (i.e., nearest neighbor of a different state). This allows us to recompute the distance values of only a few tens to hundred voxels per map update, instead of having to recompute the full dense grid of millions of voxels.

One important point about the map representation is that although Octomap allows three states (free, unknown, and occupied) with full probabilities, in order to construct a distance field, we must discretize to only two states — free and occupied. How to deal with unknown voxels is a question of safety: we cannot safely plan through them unless the robot is able to stop in known free space. Therefore, we choose to treat unknown as occupied, creating a very conservative planner. A more thorough discussion of this choice is offered in Section 6.

## 4.2 Global Planning

Next, we built a global plan to an end point in the original Octomap, while treating unknown space as free. This creates a high-level optimistic planner, while the local replanner is conservative and therefore safer. This plan will then be used as a prior for the replanner, and also allows us to use a replanner that is not complete – in case no solution is found by the local trajectory optimization, we simply stop and wait for the global planner to find a new path.

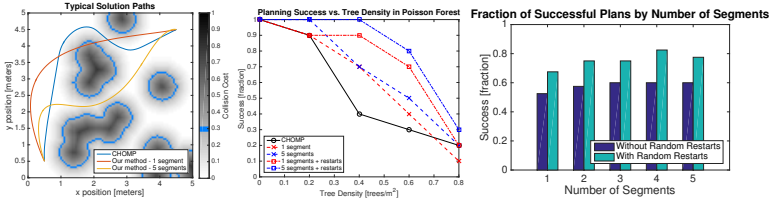
We use a 2-stage global planner: first, we find a topologically feasible straight-line path using Informed RRT* [30], and then we plan a dynamically feasible polynomial trajectory through it [8].

## 4.3 Local Replanning

To perform the local replanning, we start with the global plan (if available) or a straight-line plan to the next waypoint as prior, and incrementally update the ESDF.

We then select appropriate start and end points for the replanning algorithm. As start point, we choose the point on the current trajectory $t_R$ seconds in the future, where $t_R$ is the update rate of the replanner. Since our planner allows continuity and smoothness even in low derivatives, we are able to use the full state of the UAV at the start point, including velocity and acceleration, guaranteeing a smooth path even with changing plans.

The goal point is chosen as a point on the global trajectory that is $h$ meters ahead of the start point, where $h$ is a planning horizon. If unoccupied, we accept that




**(a)** **(b)** **(c)**

**Figure 4.2:** Results of 2D evaluations of our algorithm against CHOMP. (a): Typical paths generated by our algorithm with different number of segments, compared to CHOMP. With only one segment (red), there are not enough degrees of freedom to avoid all the obstacles, but it is able to find a solution (and one different from CHOMP) with 5 segments. The potential cost map is in gray, with the original obstacle edges in blue. (b): Success rate of the different local planners vs. density of the environment. As density increases, success rate decreases, but more so for a smaller number of segments, and some of this decrease can be counteracted by doing 10 random restarts. (c): Fraction of successful plans (taken over environments of all densities) by number of segments. This also shows a significant increase in success rate by doing random restarts, which allows the algorithm to avoid local minima that are in collision.

point as the goal, otherwise we attempt to find the nearest unoccupied neighbor in the ESDF, and as a final fallback we shorten the planning horizon until a free goal point is found.

We can then run the local optimization procedure between these two points. Either the optimization succeeds in finding a collision-free path, or we attempt random restarts until either a collision-free trajectory is found or the vehicle stops and waits for the global planner to select a new path.

## 5 Experimental Results

In this section we first evaluate the proposed continuous-time local planner and compare it to existing planning algorithms. Then, we validate our complete system in both a long, realistic simulation scenario where the robot only has local information about the environment, and then in a real-world test on an UAV avoiding newly detected objects in a room.

### 5.1 Evaluation

We validate our approach as a local start-to-goal point planner in simulation on 100 random 2D forest environments.

To analyze how our algorithm behaves in different densities of clutter, we generate $5 \times 5\,m$ Poisson forests [45] of densities between 0.2 $trees/m^2$ to 0.8 $trees/m^2$. We then analyze the success fraction of our algorithm versus CHOMP [116], a discrete-time local optimization method.

We initialize both algorithms with a straight-line path between opposite corners of the map. For our continuous-time algorithm, we use between 1 and 5 segments of 11th order polynomials ($N = 11$) minimizing snap, and optionally use 10 random restarts. For CHOMP, we use a fixed $N$ of 100 points and minimize velocity.

Fig. 4.2a shows typical paths generated by the algorithms. As can be seen, 1 segment does not have sufficient degrees of freedom to solve this planning problem, but 5 segments are able to find a short, smooth, feasible solution. CHOMP is also able to find a solution for this problem, but falls into a different local minima from our approach.

We can further analyze the behavior of the algorithms at different forest densities (number of obstacles in the environment), as seen in Fig. 4.2b. As the density of the environment increases, the chance of all methods finding a valid solution decreases. However, there is a large increase on success rate if random restarts are used, and a larger number of segments is able to handle denser environments (as in the case in Fig. 4.2a). Fig. 4.2c shows the effect of increasing the number of segments on success across all test cases.

For a more representative evaluation, we simulate arealistic 3D forest environment using real tree models, as shown in Fig. 4.3. The environments are $10 \times 10 \times 10\,m$, populated with a density of 0.2 $trees/m^2$. The trees are of random scale and height, adding the additional complexity of navigating in 3D and avoiding the tree canopies. We generate 9 such environments, and select 10 random start and goal points at least 4 meters apart, for a total of 90 test cases.

We evaluate several parameter settings of our algorithm and compare to CHOMP (which minimizes velocity), and sampling-based visibility graph search (RRT-based methods) with polynomial smoothing using 9th order minimum snap polynomials. For CHOMP and our method, since both feature a derivative cost term and a collision cost term, we use the same weights ($w_d = 0.1$, $w_c = 10$) to make as fair of a comparison as possible. Both algorithms are allowed to run for up to 50 iterations.

The results are shown in Table 4.1. Though RRT-based algorithms with smoothing are clearly able to solve a larger number of problems, Informed RRT* takes too long to run in real-time at a high rate, and RRTConnect, while significantly faster, is still exceeding the time budget and producing much longer paths.

For $N = 100$ (where $N$ is the number of discretized waypoints) in the CHOMP algorithm, the results are comparable both in run time, success rate, and path length. However, in order to fully safely verify the trajectory, there should be a waypoint for every voxel in the 3D occupancy grid. Therefore, $N = 500$ is a more appropriate comparison from a safety perspective (as the mean path length is approximately 5 meters), and since the execution time grows approximately with $O(N^2)$, this method performs much slower in such cases. It also has a lower success rate, as it does not converge to a collision-free solution within the limited iteration

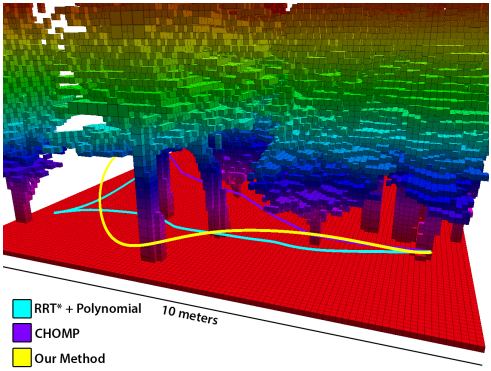



**Figure 4.3:** Figure showing the simulation setup for evaluations. The forest is $10 \times 10 \times 10$ meters, with a density of 0.2 $tree/m^2$. The paths are planned between two random points in the space, at least $4\,m$ apart. Yellow is our method, cyan is Informed RRT* + polynomial optimization (discussed in global planner section), and purple is CHOMP.

steps.

On the other hand, our method has a fixed number of parameters for a given number of segments, regardless of trajectory length or map resolution. Therefore, we are able to keep a low computation time, as long as the number of segments chosen is appropriate for the density of the environment, and our results show that 3 segments is enough for the realistic forest scenario tested. As a result, our approach has only 10 free parameters where CHOMP has 500 per axis.

We chose to use 3 segments and minimize jerk in our final real-world experiments, as this has the smallest number of free parameters for the highest success rate in our comparison.

## 5.2 System Simulation

Next, we validate our local replanning in the context of a complete system and only a partially known map.

We set up a realistic simulation experiment in RotorS [28] simulator, using a model of our multicopter platform. We approximate filling a blank map from sensor data by only giving the UAV access to a small radius of the map around itself while flying through a large forest environment. The map is $50 \times 50\,m$ with a density of 0.1 $trees/m^2$.

We use 4 meters as a planning horizon for our local replanning and give the algorithm access to 5 meters around its current position (to emulate a stereo system

| Algorithm | Success Fraction | Mean Norm. Path Length | Mean Compute Time [$s$] |
|---|---|---|---|
| Inf. RRT* + Poly | **0.9778** | **1.1946** | 2.2965 |
| RRT Connect + Poly | 0.9444 | 1.6043 | **0.5444** |
| CHOMP $N = 10$ | 0.3222 | **1.0162** | **0.0032** |
| CHOMP $N = 100$ | **0.5000** | 1.0312 | 0.0312 |
| CHOMP $N = 500$ | 0.3333 | 1.0721 | 0.5153 |
| Ours $S = 2$ jerk | 0.4889 | 1.1079 | **0.0310** |
| Ours $S = 3$ vel | 0.4778 | 1.1067 | 0.0793 |
| Ours $S = 3$ jerk | 0.5000 | 1.0996 | 0.0367 |
| Ours $S = 3$ jerk + Restart | **0.6333** | 1.1398 | 0.1724 |
| Ours $S = 3$ snap + Restart | 0.6222 | 1.1230 | 0.1573 |
| Ours $S = 3$ snap | 0.5000 | **1.0733** | 0.0379 |
| Ours $S = 4$ jerk | 0.5000 | 1.0917 | 0.0400 |
| Ours $S = 5$ jerk | 0.5000 | 1.0774 | 0.0745 |

**Table 4.1:** A table showing comparison of RRT variants with polynomial smoothing, CHOMP, and our approach on a set of 90 forest planning problems, as shown in Fig. 4.3. We compare the success fraction, normalized path length (solution path length divided by straight-line path length), and computation time. As can be seen, adding random restarts significantly improves success fraction but at the cost of higher computation time. RRT* and RRT Connect are able to solve a higher percentage of problems, but at the cost of slower performance.

with a 5 meter maximum range). A new plan, minimizing jerk in a 3-segment trajectory, is generated at 4 Hz as the UAV is flying.

Fig. 4.4 shows the results of our experiment, compared to a global plan made from a fully-known map using Informed RRT* and polynomial smoothing.

As can be seen, both algorithms produce similar paths, with the local replanning finding a solution that is only 0.5 $m$ longer than the global path. The RRT* plus smoothing algorithm ran with complete knowledge of the map and took 30 seconds to compute, 20 of which were spent on finding the visibility graph and 10 were spent on finding a collision-free polynomial path. On the other hand, our algorithm was able to find a comparable path while considering only a 4 meter region around itself and continuous replanning at 4 Hz.

## 5.3 Real World Experiments

Finally, we show our complete system running in real-time on a multicopter, starting from a completely blank map and filling it from sensor data. The experiment is done in an indoor environment with two obstacles: a large one directly in front

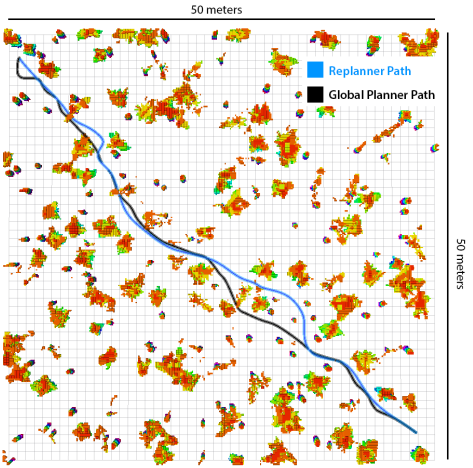



**Figure 4.4:** Here we show a comparison of our local replanning method (cyan - 67.9 $m$), operating with a planning horizon of 4 meters, compared to a single global plan (black - 67.5 $m$) generated with prior knowledge of the entire map. The environment is $50 \times 50$ $m$ with a density of $0.1\, trees/m^2$, and we show only the tree trunks of the obstacles for clarity. The local replanning algorithm is running at a rate of 4 Hz, while the global planner, using Informed RRT* with polynomial smoothing, runs in 30 seconds.

of the robot, obscuring the robot's vision, and a smaller second obstacle behind the first. A goal point is placed inside the second obstacle, and the UAV must avoid the large obstacle, fly behind it, detect the second obstacle, and stop short of collision. The physical setup is shown in Fig. 4.1.

Our platform is an Asctec Firefly[1] using a visual-inertial stereo sensor [78], running at 20 Hz, for both state estimation and perception, both of which are done entirely on-board on an 2.1 GHz Intel i7 CPU.

The UAV starts with a blank map and builds it online from dense stereo reconstruction data. The map is updated at 5 Hz, and replanning is done at 4 Hz, and we use the same parameters as in the previous section. The key difference is that due to the narrow field of view of the camera, we treated unknown space as free. A further discussion of this decision is offered in Section 6.

Fig. 4.5 shows the path evolution over time of the trajectory, with the color of the trajectory going from red to blue with time. As can be seen, though initial

[1] http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-firefly/

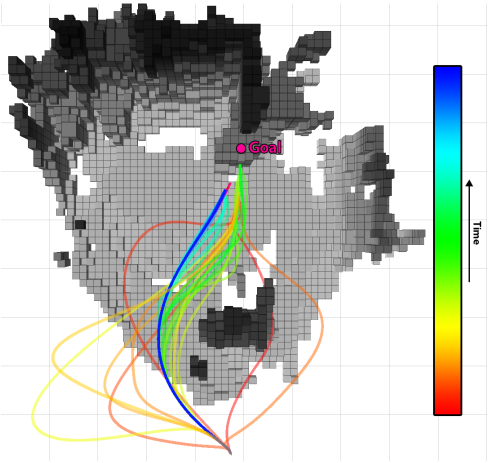



**Figure 4.5:** Local plans over time for the real-world experiments. The original goal point is embedded in the second obstacle (pink), which is not visible from the start position. Over time, the paths stop short of collision with the obstacle and the final path (blue) is a shorter, lower curvature path than many of the plans earlier in the experiment.

trajectory candidates are both in collision and often very far from obstacles, but as the UAV approaches the goal, its path gets smoother and out of collision. Also note that until the trajectory color reaches cyan, the trajectory goal is still placed in collision since the UAV has not seen the obstacle yet. This experiment can be seen in the video attachment.

We also show average timings for the complete system in Table 4.2. As can be seen, the complete system is fast enough to run in real-time at 4 Hz, and has a mean latency of only 40 ms between acquiring depth data from the sensors to generating a feasible collision-free trajectory.

# 6 Discussion

Our experiments have shown that our approach is able to find solutions to local path-planning problems successfully, at a comparable rate to existing trajectory optimization methods. While sampling-based methods are still able to solve a much larger percentage of the problems posed, they are prohibitively slow for our target application.

The main advantage of our method compared to discrete-time trajectory op-

| Step | Time [$ms$] |
|---|---|
| **Mapping** | |
| Octomap Insert | 10 |
| ESDF Initial Map Creation* | 110 |
| ESDF Incremental Update | <1 |
| **Local Replanning** | |
| Select Start and End | 1 |
| Optimization (Total) | 28 |
| Compute Der. Gradient (per 100 evals) | 0.6 |
| Computer Col. Gradient (per 100 evals) | 16 |
| **Total Time per Planner Iteration** | 40 |

**Table 4.2:** Timings for our complete replanning system, taken from the real-world experiment. We present mean timings over the entire experiment, which is why the total optimization time is shorter than the maximum time with 10 restarts (most planner iterations find a feasible solution without any restarts). Gradients timings are given over 100 evaluations of the cost function. *Initial map creation runs only once and is not included in the total.

timization methods lies in the inherently smooth, compact representation. For example, as can be seen in Table 4.1, the number of waypoints CHOMP requires to check every voxel along a 5 $m$ long path with a map resolution of 10 $cm$ is $N = 500$, and with this many variables, the convergence rate is significantly slower and the execution time is significantly longer than for our algorithm, even with 5 polynomial segments.

Our approach allows better control of end derivatives, which makes it much easier to integrate this into a continuous replanning framework, as shown in Section 5.2 and Section 5.3. We are able to continue planning from the exact current (or future) state of the UAV, leading to smooth, continuous paths. This is an advantage over both motion primitive methods, which must discretize the state, and discrete-time methods, which can only encode lower-derivative continuity as a cost rather than a hard constraint.

However, the main drawback of this approach is the required map representation (ESDF) in which space is treated as either occupied or unoccupied, and unknown space must be treated as one of the two. The obvious choice is to treat unknown space as impassable. While this can work well in simulation, real sensors often have measurements which are not completely dense, leading to blocks of unknown space even in areas that have been observed. Treating these as occupied leads to the UAV rarely being able to find areas where the entire bounding box of the UAV contains no unoccupied voxels. Treating these as free, on the other hand, encourages the UAV to travel into unknown space to avoid obstacles. This can

have disastrous consequences depending on the sensor configuration; for example, flying straight into a ceiling that the sensors can not observe. There is also the additional cost of computing a dense distance field over each voxel of the original map, which does not scale to very large environments.

However, our algorithm can be adapted to use other map representations, as long as a smooth, continuous penalty for collisions can be defined. Future work will focus on finding more compact potential cost representations without these drawbacks.

# 7 Conclusions

We presented a motion planning method that uses trajectory optimization in continuous time to find collision-free paths between obstacles. We then constructed a complete replanning system, from mapping to trajectory generation, which allows us to replan at a high rate and respond to previously unknown or unseen obstacles with low delay. We verified that our method runs comparably to discrete-time trajectory optimization, while having the advantages of continuous-time representation to minimize the number of parameters and allow arbitrary start and goal states. Our experiments showed the system running both in simulation and on a real multicopter platform at 4 Hz, though timing analysis shows that it could run at upwards of 25 Hz.

Paper II

# Signed Distance Fields: A Natural Representation for Both Mapping and Planning

Helen Oleynikova, Alex Millane, Zachary Taylor, Enric Galceran, Juan Nieto and Roland Siegwart

**Abstract**

How to represent a map of the environment is a key question of robotics. In this paper, we focus on suggesting a representation well-suited for online map building from vision-based data and online planning in 3D. We propose to combine a commonly-used representation in computer graphics and surface reconstruction, projective Truncated Signed Distance Field (TSDF), with a representation frequently used for collision checking and collision costs in planning, Euclidean Signed Distance Field (ESDF), and validate this combined approach in simulation. We argue that this type of map is better-suited for robotic applications than existing representations.

Published in:
*RSS Workshop on Geometry and Beyond*, 2016

## 1 Introduction

Any discussion of map representations in robotics generally has two sides: the perception side, which often focuses on creating high-quality 3D reconstructions of natural environments from limited sensor data, and the planning side, which uses pre-built maps to navigate through the environment in a safe and collision-free manner.

Mapping and planning often have very different requirements from an environmental representation. For mapping from the perception standpoint, it is often most important to be able to output a high-quality colored surface model, such as a mesh. For navigation and planning, it is often most essential to have fast collision checking and be able to compute clearance and direction toward nearest obstacles.

In this work, we focus on the intersection of these two fields, with special attention given to image-based sensing, such as stereo vision and RGB-D cameras: creating 3D maps, online, from noisy sensor data in order to be used by online planners for obstacle avoidance.

Euclidean Signed Distance Fields (ESDFs) have long been used in planning literature for collision checking (especially of complex shapes), inferring distances and gradients to objects for planning, and finding large free areas [116].

On the other hand, with the advent of RGB-D cameras, KinectFusion has brought projective Truncated Signed Distance Fields (TSDFs) into the forefront as a fast, flexible map representation that implicitly computes the position of the surface using zero crossings [74].

Though both of these representations are signed distance fields (SDFs), the way that the distance of a voxel is computed differs. In the case of the ESDFs, a free voxel's distance represents the Euclidean distance *to the nearest occupied voxel* (and vice-versa in the case of occupied voxels). The ESDF is computed for every voxel in the map. On the other hand, the distance of a voxel in a projective TSDF represents the distance to the surface *along the ray direction from the center of the sensor*, and is truncated to only have values very near the surface, allowing for greater compression and decreasing errors due to this approximate distance metric.

In this paper, we argue that these two approaches can be combined into one – and rather than computing separate representations for map building and planning, find a compromise that allows the same representation to be used for both.

We seek to evaluate the accuracy of ray-distance calculations versus Euclidean distance calculations given different numbers of viewpoints from a realistic vision-based sensor. We first remove the truncation requirement of the TSDF, and then evaluate how the ray distances compare to true Euclidean distances for planning purposes, and propose a hybrid approach that should retain the better qualities of both TSDFs and ESDFs for both mapping and planning.

We also offer some comparison to the most-used map representation for online 3D map building and planning in unstructured environments: Octomap [38].

The advantages of our proposed approach over map representations typically used for online planning are as follows:

- Allows a single map representation that's both well-suited for online construction from sensor data *and* usage for online planning.
- Has a more natural representation of object surface (zero-crossing of distance field), which allows easier and more accurate modeling of sensor noise.
- Separates sensor measurement (distance to surface) from measurement uncertainty (voxel weight).
- Allows fast look-up of occupancies of complex shapes, especially when represented as sets of spheres, for online planning.
- For optimization-based planning algorithms, has a natural gradient magnitude and direction both inside and outside obstacles – allowing trajectories to be 'pushed' out of collision using optimization.
- Allows extracting accurate meshed surface models for other applications.
- If desired, the same map can also be used for SLAM or state estimation, or can be built up de-coupled based on input from other state estimators.

We will cover the points in the following sections. First we will describe existing methods in both the mapping and planning fields, then discuss the advantages of using an SDF-based representation over an occupancy-based representation like Octomap, and finally provide validation for the claim that the distances used in these two representations are comparable using simulation.

# 2 Related Work

This section discusses relevant previous work and details of the approaches in mapping and planning literature.

## 2.1 Mapping Literature

SDFs have long been used for representing 3D volumes in computer graphics [31] [27]. They have also been used to build offline reconstructions of objects from real sensor data since the 1990s [16].

However, TSDFs have come back into the forefront of computer vision and robotics in 2011 with the new RGB-D Kinect sensors and the work from Newcombe *et al.* on KinectFusion [74]. Their approach focuses on doing high-resolution, accurate 3D reconstructions from RGB-D data on a GPU in real-time, using a TSDF as the main representation. They provide a Simultaneous Localization and Mapping (SLAM) approach, which estimates the pose of the camera at the same time as creating the reconstruction.

There have been a number of extensions to this approach, including Kintinuous, which allows scanning much larger spaces [112], and FastFusion which allows online reconstruction on the CPU rather than GPU [104] in an octree-style voxel grid. The main end-result of these is to generate a high-fidelity mesh, usually using a marching cubes algorithm [61].

Another competing representation for high-resolution online mapping from RGB-D data is RGB-D SLAM [35], which uses surfels (small planar units with size, color,

and surface normal) to represent 3D structure, as a volumetric analogue to sparse pointclouds. Such representations are able to take advantages of the SLAM techniques developed for sparse keypoint-based maps and are better suited to distorting geometry, for example in ElasticFusion [113] where the map is distorted when encountering loop closures. These methods generally have high accuracy for state estimation and high-quality models. However, Bylow *et al.* have shown that it is possible to have the same level of accuracy from TSDF-based maps [11].

Therefore, SDF-based representations are well-studied, fast, and accurate for online surface reconstruction. In the following section, we will discuss how SDF-based representations are used for planning.

## 2.2 Planning Literature

Maps are essential to planning collision-free paths, with the representation of the map defining both the quality of the resulting path, and also what kind of planning algorithms can be used.

The minimum amount of information a map must provide is occupancy of a given point in space. Assuming a fixed-size grid, this enables the use of many different classes of planning algorithm: search-based methods like A* and D*-Lite [47], sampling-based methods like RRTs [53].

Occupancy grids represent the most commonly-used type of map representation for planning in 2D [20]. The approach of Elfes *et al.* is to use a fixed-size grid, probabilistic model of sensor measurements and model observed (known) and unknown space explicitly, allowing the incorporation of complex sensor models and reasoning about the environment. There are now many options available off-the-shelf that will run a complete 2D SLAM system online and provide occupancy grids of previously unknown environments [65], and countless planning algorithms that take 2D occupancy grids as input [26].

Naively extending occupancy grids to 3D, however, leads to huge memory requirements as well as slow ray-casts and look-ups for any space larger than a room. The solution most commonly used in 3D contexts while building a map online is Octomap [38]. This approach uses an octree-based representation of occupancy probabilities of cells in 3D space. The octree structure allows large blocks of space with the same probability to be represented by a single large cell, therefore vastly decreasing the amount of memory needed to represent areas of unknown or free space.

However, there are planning approaches which require additional information from a map. For example, trajectory optimization-based planners, such as CHOMP [116] and TrajOpt [99] require the distance to obstacles and occupancy gradient information. Algorithms such as these require an ESDF that is not truncated, and contains distance values over the entire voxel space. Usually these are constructed from another map representation, and often from a map hand-crafted out of object primitives (spheres, cubes) or high-fidelity mesh models of objects for manipulation [87].

Having a distance map also speeds up collision checking of complex shapes – for

example, many-jointed robot arms are commonly represented as a set of overlapping spheres and check the distance field in the center of each sphere (which is one look-up in the ESDF per sphere) [115] [116] [40].

For gradient-based trajectory optimization methods, the collision cost (which is necessary to produce collision-free trajectories) also needs a gradient. For these, the ESDF gives a natural cost (a function, such as hinge loss [99] or a smoothed hinge loss [116] of the distance) and checking the distance values of the neighbors gives the gradient at a given point. This allows CHOMP and other such methods to follow the upward gradient of the distance to push points on the trajectory out of collision.

Wanger *et al* [110] is the closest work to our proposed approach, where a complete ESDF is built from the output of KinectFusion [74] and used for trajectory planning with CHOMP. However, their approach builds the entire ESDF at once from TSDF data, while our work focuses on exploring ways to combine these two representations into one.

In summary, SDFs allow for faster collision checking than occupancy grids while providing additional data needed for optimization-based planning methods.

## 3 SDF Advantages over Octomap

In this section, we make arguments about why using an SDF is a better map representation for both perception (creating the map) and planning (using the map for collision avoidance and clearance calculations) than the most commonly used 3D representation, the Octomap [38].

Since its advent, Octomap has been very widely used for 3D robotics applications, most notably for UAVs [34], [8]. We believe that this is due to a number of factors: first and not least, the open-source implementation and associated ROS wrappers have made it a very easy off-the-shelf solution for many applications. Second, the 'probabilistic' nature of the representation (assigning probabilities to each raycast, merging multiple observations of the same scene together) make it a good representation for noisy sensor data, such as stereo matching or RGB-D sensors where 'speckles' are common. This adds a level of low-pass filtering even to sensors exhibiting non-Gaussian error models. The third is due to memory efficiency and speed: the flexible voxel size allows representing large areas, and with some straight-forward optimizations, it is possible to get the insertion time of a dense stereo scan at 320x240 down to approximately 10 ms, and performing collision checks (even for a large bounding box) in this space is also very fast [34].

However, this representation also has downsides. The first is that the probability model used does does not accurately represent the error model of vision-based depth sensing. Since Octomap was originally designed to use with laser measurements, the accuracy of which does not degrade with distance to the sensor, the Octomap sensor model has a single probability of occupancy for one voxel at the end of the ray-cast. However, this is not an accurate model for stereo- or other vision-based sensing, where it is possible to have an expected error of over a meter

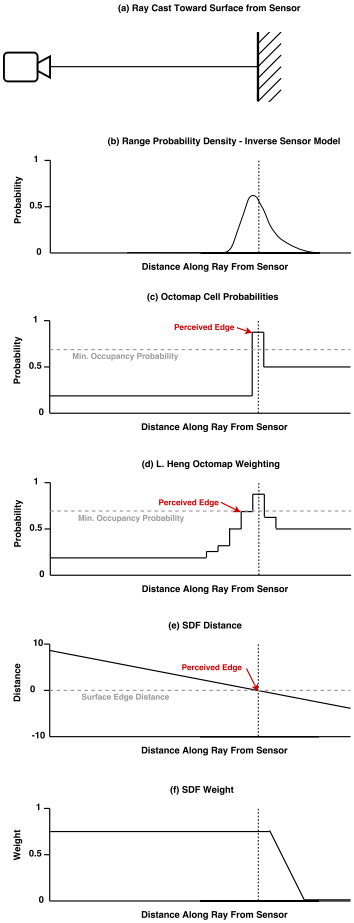



**Figure 5.1:** Comparison of results from a single ray-cast in the various representations discussed: (a) diagram of a single vision-based sensor ray hitting a surface, (b) probability for range from an inverted stereo sensor model [36], (c) vanilla Octomap [38] probabilities of the raycast, (d) L. Heng Weighing [34], (e) TSDF (truncation radius not shown) [74], and (f) the TSDF weights along this ray [11]. Note that these are illustrations, and not to scale.

at high distances, depending on the camera setup [36]. Heng *et al* implement a more realistic sensor model using distance weighing in the Octomap, however this tends to have the effect of inflating obstacles in the map [33].

Fig. 5.1 shows a representation of the different weighing representations for a single 1D ray within Octomap, compared to a TSDF representation. Fig. 5.1a shows a diagram of a ray cast from a depth camera hitting a wall, Fig. 5.1b shows the inverse sensor model for a stereo camera observing that wall, and Fig. 5.1c-d show vanilla and L. Heng weighing for the sensor hit. Since Octomap has a discrete cut-off probability for considering space occupied, the figure shows why L. Heng weighting tends to distort or inflate object boundaries.

One advantage of the TSDF is that even when discretized, it models a continuous function, as shown in Fig. 5.1e. Therefore it is possible to recover the position of the surface at a precision above the minimum voxel size, allowing the use of larger voxels and therefore smaller maps in memory.

The other advantage over Octomap is that TSDF has two values for each voxel: the distance to the surface (along the ray from the camera) and the weight/probability of this measurement. This allows us to more accurately model the actual error of vision-based depth estimates, and when merging multiple measurements, leads to a maximum-likelihood estimate of the surface, since the surface is found as a zero crossing. Bylow *et al* evaluate different weighing functions for TSDFs [11], and we show the linear weighing used by KinectFusion in Fig. 5.1f. However, since this value is separate from the actual distance measurement, any model can be incorporated without necessarily inflating the surface.

While the advantages for perception are clear, an SDF-based implementation has advantages in terms of path planning as well. Here we discuss the advantages of an ESDF (using Euclidean distances to nearest occupied/unoccupied space) over a binary occupancy-based representation. We will discuss how we can combine the TSDF and ESDF in the following section.

As discussed in Section 2.2, an ESDF allows fast collision checks for complex shapes, as long as they can be expressed as a set of overlapping spheres. It also permits using gradient-based methods, as it gives a smooth cost of collision, which decreases as an object approaches free space. This also allows computation of collision cost gradients, and therefore choosing directions which lead to decreasing costs.

## 4 Combining ESDF and TSDF: Results

The main difference between the two representations, TSDF for mapping and ESDF for planning, is the way that distances are computed. Both are signed distance fields, as the names suggest, but the distance in each voxel represents a different quantity.

In the ESDF, the distance in each voxel is the Euclidean distance *to the nearest occupied cell* (or if inside an object, distance to the nearest free cell). In the TSDF, on the other hand, the distance is computed *along the sensor ray* – that

is, it represents a distance to the nearest occupied cell not in Euclidean space, but along this one-dimensional ray extending from the sensor center.

Here we present models showing how different ray distances (TSDF) are versus Euclidean distances (ESDF) and how this affects the quality of the map for planning. We focus on evaluating the metrics that are important for planning, especially for local optimization-based planners: error in distance to obstacles and direction of the gradient of the field.

We suggest four strategies to evaluate. The first two are building a projecting TSDF without a minimum truncation distance with two different strategies for merging multiple scans into the map:

1. Average the sensor value with the map value, giving equal weight to both (average weighing)
2. Take the minimum distance outside obstacles, and the maximum distance inside (minimum weighing)

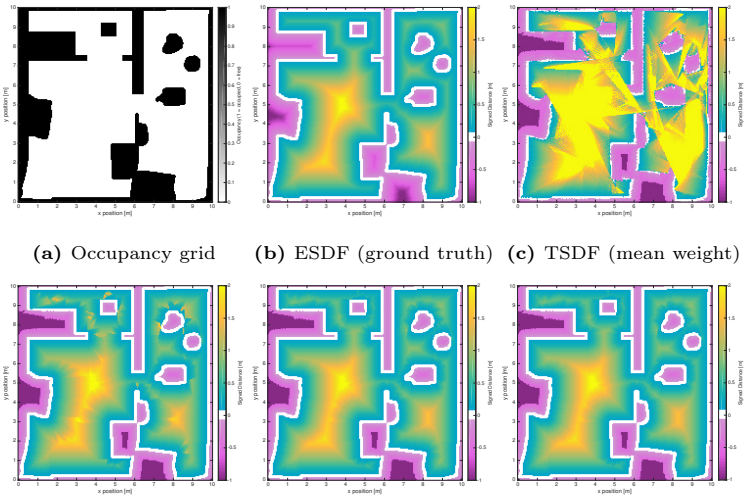
We also propose a hybrid E/TSDF, where outside some minimum truncation distance, we iteratively compute Euclidean distances based on the projective distances inside the truncation radius with every new integrated scan. Finally, we also present the 'standard' approach for comparison, where we compute an occupancy grid and calculate Euclidean distances from that grid after building the complete map.

We use a vastly simplified model of the TSDF in 2D and a simulated sensor with a 70° field of view, 0.5° angular resolution, and a maximum range of $8m$. There is no noise on this sensor, but it is discretized to the voxel size. The sample environment attempts to mimic an enclosed office space of $10m \times 10m$ with many occlusions, some geometric and some organic shapes. Fig. 5.2a shows the occupancy map we use for the simulations and Fig. 5.2b shows the ground-truth ESDF computed from this known map, where white is the object borders, pink is inside the objects, and green and yellow are distances (yellow being the furthest from an obstacle).

We generated random viewpoints within this space, sampling uniformly from unoccupied space and yaw. We then cast rays into this map, and generated distance measurements until $0.5m$ behind a surface. When raycasting into unknown space, the value was simply updated to the ray measurement value. When raycasting into space that has been observed before, we use the strategies described above.

Fig. 5.2c shows the TSDF generated with average weighing, and Fig. 5.2d shows the results of minimum weighing after 500 viewpoints. As can be seen, average weighing always overestimates the distance – as ray-distance is always greater than or equal to true Euclidean distance (it is equal in the case where the ray direction is along the surface normal, and greater in every other case). Whereas minimum weighing, over a large number of viewpoints, starts to converge to the true ESDF.

Of course, with fewer viewpoints, the estimate is worse – Table 5.1 shows the mean absolute error of distance measurements (taken only outside obstacles) over all observed voxels for different number of viewpoints. $r$ is the radius around the surface at which errors are evaluated; $r = 0.5m$ is an estimate of the errors close to the surface, and $r = \infty$ is evaluated over the whole map. The values are compared to the ground-truth ESDF shown in Fig. 5.2b. However, this is not a strictly fair



**(a)** Occupancy grid **(b)** ESDF (ground truth) **(c)** TSDF (mean weight)

**(d)** TSDF (min. weight) **(e)** Hybrid T/ESDF **(f)** Occupancy + ESDF

**Figure 5.2:** Results of various weighing schemes in SDF, using 500 viewpoints. White represents the observed surface of the object (zero crossings), green and yellow are areas outside the object, and pink are areas inside the object. Quantitative comparison is presented in Table 5.1.

comparison, as all the viewpoint generated maps have discretization errors and not all obstacles are observed (for a lower number of viewpoints). We therefore also present the results for "Occupancy + ESDF", which was generated by creating a standard occupancy map from the sensor measurements and computing an ESDF from that map. It can be thought of as a lower bound on the error (though in some cases it actually has slightly higher error than other representations, which is due to discretization).

The other method we evaluate is a hybrid T/ESDF, which functions by behaving like a TSDF around surface edges, holding this surface section fixed and iteratively updating all other values using Euclidean distances. This retains the desirable quantities of both representations – near the surface for mapping and surface reconstruction, and further away from the surface for planning, while adding only slight computation cost per new viewpoint. Table 5.1 shows that this hybrid approach has the lowest errors of the TSDF-based approaches, and is comparable to the ESDF map built from occupancy grids.

This shows that depending on the weighing scheme, with sufficient distinct view-

| Viewpoints | TSDF Average Weighing | | TSDF Min Weighing | | Hybrid E/TSDF | | Occupancy + ESDF | | Unobserved Ratio |
|---|---|---|---|---|---|---|---|---|---|
| **Mean Abs. Error** $[m]$ | r = $0.5m$ | r = $\infty$ | r = $0.5m$ | r = $\infty$ | r = $0.5m$ | r = $\infty$ | r = $0.5m$ | r = $\infty$ | |
| V = 10 | 0.699686 | 1.022263 | 0.501408 | 0.854449 | **0.134561** | **0.480509** | 0.162672 | 0.541797 | 0.659258 |
| V = 50 | 0.730469 | 1.009407 | 0.317534 | 0.374416 | **0.067819** | **0.078796** | 0.070928 | 0.086687 | 0.054670 |
| V = 100 | 0.717782 | 0.972221 | 0.157599 | 0.197076 | 0.042314 | 0.051344 | **0.037105** | **0.049586** | 0.010183 |
| V = 500 | 0.628532 | 0.788858 | 0.031576 | 0.031498 | 0.015214 | 0.014252 | **0.009491** | **0.010818** | 0.000000 |
| **Gradient Error Mag.** $[m]$ | 0.198165 | | 0.014947 | | 0.030057 | | **0.004643** | | |
| **Gradient Error Ang.** [°] | **6.065860** | | 8.699339 | | 9.735610 | | 8.699339 | | |

**Table 5.1:** Error analysis of different TSDF representations compared to ESDF. Results of the simulating a 2D SDF with a realistic sensor, comparing the error in using the TSDF distance (distance along the sensor ray) and comparing the error to the true ESDF (distance from nearest object). $r$ is the radius around the surface at which errors are evaluated: $r = 0.5$ only evalutes distances close to the surface, while $r = \infty$ evaluates all distances in the map. Since below 500 viewpoints, not all parts of the map have been observed, we also give a Unobserved Ratio for reference.

points, ray-distance approximates Euclidean distance, and there exist hybrid approaches that can further increase the accuracy of these combined maps.

## 5 Conclusions

In this paper, we compare two signed distance field representations: truncated signed distance fields (TSDFs), used for computer graphics and surface reconstruction from depth data, and Euclidean signed distance fields (ESDFs), used in planning for fast collision checking and cost and gradient information for optimization-based path planners.

We show the advantages of SDF-based maps for online map building and online planning in 3D compared to the commonly-used Octomap representation [38] and validate some of our claims by showing that projective ray-distances (used in TSDFs) can approximate Euclidean distances (used in ESDFs) when using sufficient viewpoints and an intelligent merging strategy. We also propose a hybrid approach which has advantages of both ESDFs and TSDFs.

We hope that this work can be a starting point for considering different map representations for online mapping for planning and navigation in 3D.

Paper III

# Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning

Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto

**Abstract**

Micro Aerial Vehicles (MAVs) that operate in unstructured, unexplored environments require fast and flexible local planning, which can replan when new parts of the map are explored. Trajectory optimization methods fulfill these needs, but require obstacle distance information, which can be given by Euclidean Signed Distance Fields (ESDFs).

We propose a method to incrementally build ESDFs from Truncated Signed Distance Fields (TSDFs), a common implicit surface representation used in computer graphics and vision. TSDFs are fast to build and smooth out sensor noise over many observations, and are designed to produce surface meshes.

We show that we can build TSDFs faster than Octomaps, and that it is more accurate to build ESDFs out of TSDFs than occupancy maps. Our complete system, called voxblox, is available as open source and runs in real-time on a single CPU core. We validate our approach on-board an MAV, by using our system with a trajectory optimization local planner, entirely on-board and in real-time.

Published in:
*IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017
DOI: 10.1109/IROS.2017.8202315

## 1 Introduction

Rotary-wing Micro Aerial Vehicles (MAVs) have become one of the most popular robotics research platforms, as their agility and small size makes them ideal for many inspection and exploration applications. However, their low payload and power budget, combined with fast dynamics, requires fast and light-weight algorithms. Planning in unstructured, unexplored environments poses a particularly difficult problem, as both mapping and planning have to be done in real-time. In this work, we focus specifically on providing a map for local planning, which quickly finds feasible paths through changing or newly-explored environments. Furthermore, humans often supervise high-level mission goals, and therefore we also aim to provide a human-readable representation of the environment.

While many algorithms are well-suited for global MAV planning (such as RRTs, graph search methods, and mixed-integer convex programs), local re-planning requires algorithms that can find feasible (though not necessarily optimal) paths in minimal time. Trajectory optimization-based planning methods are well suited to these problems, as they are very fast and able to deal with complex environments. However, they require the distances to obstacles to be known at all points in a map, as well as distance gradients [81, 93]. These distance maps are usually computed from an occupancy map such as Octomap [38], most often in batch, but more recently some incremental approaches have appeared [51]. The main drawback of these methods is that the maximum size of the map must be known *a priori*, and cannot be dynamically changed.

We attempt to overcome these shortcomings by proposing a system capable of incrementally building Euclidean Signed Distance Fields (ESDFs) online, in real-time on a dynamically growing map, while using an underlying map representation that is well-suited to visualization. ESDFs are a voxel grid where every point contains its *Euclidean* distance to the nearest obstacle. Truncated Signed Distance Fields (TSDFs) have recently become a common implicit surface representation for computer graphics and vision applications [16, 74], as they are fast to construct, filter out sensor noise, and can create human-readable meshes with sub-voxel resolution. In contrast to ESDFs, they use *projective* distance, which is the distance along the sensor ray to the measured surface, and calculate these distances only within a short *truncation radius* around the surface boundary. We propose to build ESDFs directly out of TSDFs and leverage the distance information already contained within the truncation radius, while also creating meshes for remote human operators. We assume that the MAV is using stereo or RGB-D as the input to the map, and that its pose estimate is available.

Our experiments on real datasets show that we can build TSDFs faster than Octomaps [38], which are commonly used for MAV planning. We also analyze sources of error in our ESDF construction strategy, and validate the speed and accuracy of our method against simulated ground truth data. Based on these results, we make recommendations on the best parameters for building both TSDFs and ESDFs for planning applications. Finally, we show the complete system integrated and running in closed-loop as part of an online replanning strategy, entirely on-board

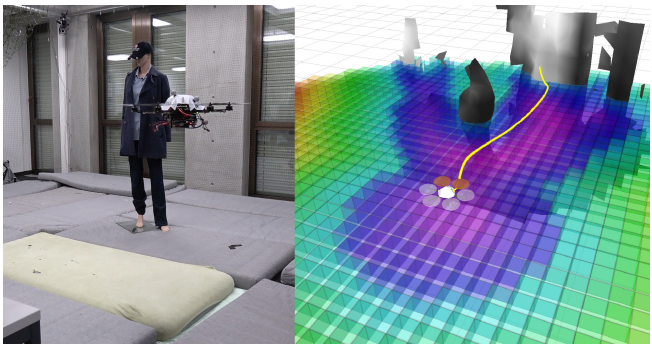


**Figure 6.1:** A planning experiment using voxblox-generated TSDF (shown as grayscale mesh) and ESDF (shown as a horizontal slice of the 3D grid) running entirely in real-time and on-board the MAV, not using any external sensing. The vehicle attempts to plan to a point behind the mannequin by using a trajectory optimization-based method [81] which relies on having smooth distance costs and gradients.

an MAV. This complete system, named **voxblox**, is available as an open-source library at `github.com/ethz-asl/voxblox`.

The contributions of this work are as follows:

- Present the first method to incrementally build ESDFs out of TSDFs in dynamically growing maps.
- Analyze different methods of building a TSDF to maximize reconstruction speed and surface accuracy at large voxel sizes.
- Provide both analytical and experimental analysis of errors in the final ESDF, and propose safety margins to overcome these errors.
- Validate the complete system by performing online replanning using these maps on-board an MAV.

## 2 Related Work

This section gives a brief overview of different map representations used for planning, and existing work in building ESDFs and TSDFs.

Occupancy maps are a common representation for planning. One of the most popular 3D occupancy maps is called Octomap [38], which uses a hierarchical octree structure to store occupancy probabilities. However, there are planning approaches for which only occupancy information is insufficient. For example, trajectory optimization-based planners, such as CHOMP [116], require distances

to obstacles and collision gradient information over the entire workspace of the robot. This is usually obtained by building an ESDF in batch from another map representation.

While creating ESDFs or Euclidean Distance Transforms (EDTs) of 2D and 3D occupancy information is a well-studied problem especially in computer graphics, most recent work has focused on speeding up batch computations using GPUs [12, 107]. However, our focus is to minimize computation cost on a CPU-only platform.

Lau *et al.* have presented an efficient method of incrementally building ESDFs out of occupancy maps [51]. Their method exploits the fact that sensors usually observe only a small section of the environment at a time, and significantly outperforms batch ESDF building strategies for robotic applications. We extend their approach to be able to build ESDFs directly out of TSDFs, rather than from occupancy data, exploiting the existing distance information in the TSDF.

TSDFs, originally used as an implicit 3D volume representation for graphics, have become a popular tool in 3D reconstruction with KinectFusion [74], which uses the RGB-D data from a Kinect sensor and a GPU adaptation of Curless and Levoy's work [16], to create a system that can reconstruct small environments in real-time at millimeter resolution.

The main restriction of this approach is the fixed-size voxel grid, which requires a known map size and a large amount of memory. There have been multiple extensions to overcome this shortcoming, including using a moving fixed-size TSDF volume and meshing voxels exiting this volume [112], using an octree-based voxel grid [104], and allocating blocks of fixed size on demand in a method called *voxel hashing* [76]. We follow the voxel-hashing approach to allow our map to grow dynamically as the robot explores the environment.

The focus of all of these methods is to output a high-resolution mesh in real-time using marching cubes [61], frequently on GPUs. There has also been work on speeding up these algorithms to run on CPU [104] and even on mobile devices [46]; however, the application of high-resolution 3D reconstruction remains the same. Instead, our work focuses on creating representations that are accurate and fast enough to use for planning onboard mobile robots, while using large voxels to speed up computations and save memory.

One existing work that combines ESDFs and TSDFs is that of Wagner *et al.*, who use KinectFusion combined with CHOMP for planning for an armed robot [110, 111]. However, instead of updating the ESDF incrementally, they first build a complete TSDF, then convert it to an occupancy grid and compute the ESDF in a single batch operation for a fixed-size volume. In contrast, our incremental approach gives us the ability to maintain an ESDF directly from a TSDF, handle dynamically growing the map without knowing its size *a priori*, and is significantly faster than batch methods.

Features such as CPU computation time, incremental ESDF construction, and dynamically-growing map are essential for a map representation to use for on-board local planning for an MAV.

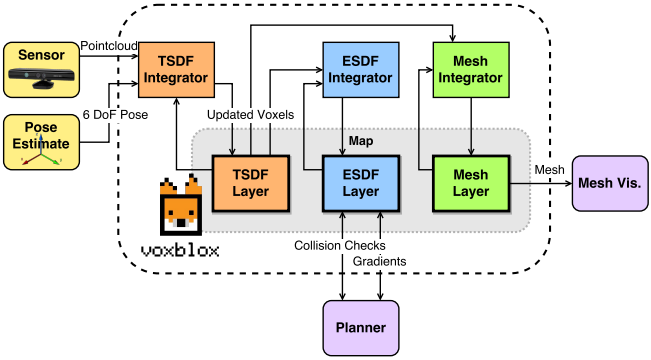



**Figure 6.2:** System diagram for voxblox, showing how the multiple map layers (TSDF, ESDF, and mesh) interact with each other and with incoming sensor data through integrators.

## 3 System

Our overall system functions in two parts: first, incorporating incoming sensor data into a TSDF (described in detail in Section 4), and then propagating updated voxels from the TSDF to update the ESDF (see Section 5).

Fig. 6.2 shows the overall system diagram. Sensor data from stereo or RGB-D sensors comes in as colored pointclouds, which are then integrated into the TSDF as discussed in Section 4 using raycasting into the voxel map. Updated voxels from the TSDFs are marked, and then at a given frequency, the ESDF is updated by propagating changes from the TSDF and doing wavefront propagation, as shown in Section 5. The mesh is also built on-demand from the latest state of the TSDF for visualization purposes.

In order to make it suitable for exploration and mapping applications, we use a dynamically sized map that makes use of the voxel hashing approach of Niessner *et al.* [76]. Each type of voxel (TSDF or ESDF) has its own layer, and each layer contains independent blocks that are indexed by their position in the map. A mapping between the block positions and their locations in memory is stored in a hash table, allowing $\mathcal{O}(1)$ insertions and look-ups. This makes the data structure flexible to growing maps, and additionally allows faster access than octree structures such as used by Octomap (which is $\mathcal{O}(\log n)$).

## 4 TSDF Construction

TSDFs are constructed out of pointcloud data by raycasting points in a sensor pointcloud into a global map, then averaging the new projective distance measure-

ments into existing voxels, calculating distances only up to a truncation distance of $\delta$. Choices in how to build a TSDF out of sensor data can have a large impact on both the integration speed and the accuracy of the resulting reconstruction. Here we present weighting (how new measurements are averaged with existing measurements) and merging (how points from the sensor data are grouped) strategies, which increase accuracy and speed especially at large voxel sizes.

## 4.1 Weighting

A common strategy to integrate a new scan into a TSDF is to ray-cast from the sensor origin to every point in the sensor data, and update the distance and weight estimates of voxels along this ray. The choice of weighting function can have a strong impact on the accuracy of the resulting reconstruction, especially for large voxels, where thousands of points may be merged into the same voxel *per scan*.

KinectFusion discussed using weights based on $\theta$, the angle between the ray from the sensor origin and the normal of the surface, however advocate for using a simpler constant weight [74]. This is a common approach in other literature [11, 39, 76, 112].

The general equations governing the merging are based on the existing distance and weight values of a voxel, $D$ and $W$, and the new update values from a specific point observation in the sensor, $d$ and $w$, where $d$ is the distance from the *surface boundary*. Given that $\mathbf{x}$ is the center position of the current voxel, $\mathbf{p}$ is the position of a 3D point in the incoming sensor data, $\mathbf{s}$ is the sensor origin, and $\mathbf{x}, \mathbf{p}, \mathbf{s} \in \mathbb{R}^3$, the updated $D$ distance and $W$ weight values of a voxel at $\mathbf{x}$ will be:

$$d(\mathbf{x}, \mathbf{p}, \mathbf{s}) = \|\mathbf{p} - \mathbf{x}\| \operatorname{sign}\big((\mathbf{p} - \mathbf{x}) \bullet (\mathbf{p} - \mathbf{s})\big) \tag{6.1}$$

$$w_{\text{const}}(\mathbf{x}, \mathbf{p}) = 1 \tag{6.2}$$

$$D_{i+1}(\mathbf{x}, \mathbf{p}, \mathbf{s}) = \frac{W_i(\mathbf{x}) D_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p}) d(\mathbf{x}, \mathbf{p}, \mathbf{s})}{W_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p})} \tag{6.3}$$

$$W_{i+1}(\mathbf{x}, \mathbf{p}) = \min\big(W_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p}), W_{\max}\big) \tag{6.4}$$

We propose a more sophisticated weight to compare to the constant weighting shown above. Bylow *et al.* compared the effects of dropping the weight off behind the isosurface boundary, and found that a linear drop-off often yielded the best results [11]. Nguyen *et al.* empirically determined the RGB-D sensor model, and found that the $\sigma$ of a single ray measurement varied predominantly with $z^2$ [75], where $z$ is the depth of the measurement in the camera frame. We combined a simplified approximation of the RGB-D model with the behind-surface drop-off as follows:

$$w_{\text{quad}}(\mathbf{x}, \mathbf{p}) = \begin{cases} \frac{1}{z^2} & -\epsilon < d \\ \frac{1}{z^2} \frac{1}{\delta - \epsilon}(d + \delta) & -\delta < d < -\epsilon \\ 0 & d < -\delta, \end{cases} \tag{6.5}$$

where we use a truncation distance of $\delta = 4v$ and $\epsilon = v$, and $v$ is the voxel size.

Intuitively, this would make an even bigger difference in the presence of thin surfaces that are observed from multiple viewpoints, as this reduces the influence of voxels that have actually not been directly observed (those behind the surface). An analysis of the effect this weighting has on surface reconstruction accuracy is presented in Section 6.1.

## 4.2 Merging

We aim to speed up merging of new sensor data into the TSDF by designing a strategy that only performs raycasts once per end voxel, exploiting the relatively large voxel size compared to the resolution of the incoming sensor pointclouds.

There are two main methods for integrating information from a sensor data into a TSDF: raycasting [16] and projection mapping [74] [46].

Raycasting casts a ray from the camera optical center to the center of each observed point, and updates all voxels from the center to truncation distance $\delta$ behind the point. Projection mapping instead projects voxels in the visual field-of-view into the depth image, and computes its distance from the distance between the voxel center and the depth value in the image. It is significantly faster, but leads to strong aliasing effects for larger voxels [46].

Our approach, *grouped raycasting*, significantly speeds up raycasting without losing much accuracy. For each point in the sensor scan, we project its position to the voxel grid, and group it with all other points mapping to the same voxel, taking the mean color and distance across grouped points and performing raycasting only once. This leads to a very similar reconstruction result while being up to 20 times faster than the naive raycasting approach, as shown in Section 6.1.

# 5 Constructing ESDF from TSDF

In this section we discuss how to build an ESDF for planning out of a TSDF built from sensor data, and then analyze bounds on the errors introduced by our approximations.

## 5.1 Construction

We base our approach on the work of Lau *et al.*, who present a fast algorithm for dynamically updating ESDFs from occupacy maps [51]. We extend their method to take advantage of TSDFs as input data, and additionally allow the ESDF map to dynamically change size. The complete method is shown in Algorithm 1, where $v_{\mathrm{T}}$ represents a voxel in the original TSDF map and $v_{\mathrm{E}}$ is the co-located voxel in the ESDF map.

One of the key improvements we have made is to use the distance stored in the TSDF map, rather than computing the distance to the nearest occupied voxel. In the original implementation, each voxel had an occupied or free status that the algorithm could not change. Instead, we replace this concept with a *fixed* band around the surface: ESDF voxels that take their values from their co-located TSDF

voxels, and may not be modified. The size of the fixed band is defined by TSDF voxels whose distances fulfill $|v_{\mathrm{T}}.d| < \gamma$, where $\gamma$ is the radius of the band, analyzed further in Section 5.2.

The general algorithm is based on the idea of *wavefronts* – waves that propagate from a start voxel to its neighbors (using 26-connectivity), updating their distances, and putting updated voxels into the wavefront queue to further propagate to their neighbors. We use two wavefronts: raise and lower. A voxel gets added to the raise queue when its new distance value from the TSDF is higher than the previous value stored in the ESDF voxel. This means the voxel, and all its children, need to be invalidated. The wavefront propagates until no voxels are left with parents that have been invalidated.

The lower wavefront starts when a new fixed voxel enters the map, or a previously observed voxel lowers its value. The distances of neighboring voxels get updated based on neighbor voxels and their distances to the current voxel. The wavefront ends when there are no voxels left whose distance could decrease from its neighbors.

Unlike Lau *et al.* [51], who intersperse the lower and raise wavefronts, we raise all voxels first, then lower all voxels to reduce bookkeeping. Additionally, where they treat unknown voxels as occupied, we do not update unknown voxels. For each voxel, we store the direction toward the parent, rather than the full index of the parent. For quasi-Euclidean distance (shown in the algorithm), this parent direction is toward an adjacent voxel, while for Euclidean distance, it contains the full distance to the parent. A full discussion of Euclidean versus quasi-Euclidean distance is offered in the section below.

Finally, since new voxels may enter the map at any time, each ESDF voxel keeps track of whether it has already been observed. We then use this in line 20 of Algorithm 1 to do a crucial part of bookkeeping for new voxels: adding all of their neighbors into the *lower* queue, so that the new voxel will be updated to a valid value.

Our approach incorporates a bucketed priority queue to keep track of which voxels need updates, with a priority of $|d|$. In the results, we compare two different variants: a FIFO queue and a priority queue (where the voxel with the smallest absolute distance is updated first).

## 5.2 Sources of Error in ESDF

When using maps for planning, it is essential to know what effect the method has on the error in the final distance computations. In this section, we aim to quantify the effect of our approximations and recommend a safety margin by which to increase bounding boxes used for planning.

We consider two key contributions to error in the final ESDF: first, the TSDF projective distance calculations, and second, the quasi-Euclidean approximation in distance calculations.

Projective distance (distance along the camera ray to the surface) will always match or overestimate the actual Euclidean distance to the nearest surface. Therefore, to use projective distances from the TSDF, we need to quantify the error this

**Algorithm 1** Updating ESDF from TSDF

```
 1: function Propagate(map_ESDF, map_TSDF)
 2:     for each voxel v_T in updated voxels in map_TSDF
 3:         if isFixed(v_T)
 4:             if v_E.d > v_T.d  or not v_E.observed
 5:                 v_E.observed ← True
 6:                 v_E.d ← v_T.d
 7:                 Insert(lower, v_E)
 8:             else
 9:                 v_E.d ← v_T.d
10:                 Insert(raise, v_E)
11:                 Insert(lower, v_E)
12:         else
13:             if v_E.fixed
14:                 v_E.observed ← True
15:                 v_E.d ← sign(v_T.d) · d_max
16:                 Insert(raise, v_E)
17:             else if not v_E.observed
18:                 v_E.observed ← True
19:                 v_E.d ← sign(v_T.d) · d_max
20:                 InsertNeighbors(lower, v_E)
21:     ProcessRaiseQueue(raise)
22:     ProcessLowerQueue(lower)
23: function isFixed(v_E) return −γ < v_E.d < γ
24: function InsertNeighbors(queue, v_E)
25:     for each neighbor of v_E
26:         Insert(queue, neighbor)
27: function ProcessRaiseQueue(raise)
28:     while raise ≠ ∅
29:         v_E ← pop(raise)
30:         v_E.d ← sign(v_E.d) · d_max
31:         for each neighbor of v_E
32:             if v_E.direction(neighbor) = neighbor.parent
33:                 Insert(raise, neighbor)
34:             else
35:                 Insert(lower, neighbor)
36: function ProcessLowerQueue(lower)
37:     while lower ≠ ∅
38:         v_E ← pop(lower)
39:         for each neighbor of v_E at distance dist
40:             if neighbor.d > 0 and v_E.d + dist < neighbor.d
41:                 neighbor.d ← v_E.d + dist
42:                 neighbor.parent ← −v_E.direction(neighbor)
43:                 Insert(lower, neighbor)
44:             else if neighbor.d < 0 and v_E.d − dist > neighbor.d
45:                 neighbor.d ← v_E.d − dist
46:                 neighbor.parent ← −v_E.direction(neighbor)
47:                 Insert(lower, neighbor)
```

will introduce. The error is dependent on $d$, the measured distance of the voxel, and $\theta$, the incidence angle between the camera ray and the object surface. We assume locally planar objects. The projective error residual $r_p(\theta)$ can therefore be expressed as:

$$r_p(\theta) = d\sin(\theta) - d \tag{6.6}$$

For the purposes of this analysis, we assume that the incidence angle $\theta$ can be between $\pi/20$ and $\pi/2$, and is uniformly distributed in this range. The lower bound of $\pi/20$ comes from the observation that a camera ray can not be parallel to a surface boundary, nor can a camera exist infinitesimally close to a surface, due to the physical dimensions of the camera. $\pi/20$ corresponds to an MAV a minimum of 1 meter away from a surface with a maximum sensor ray length of 5 meters. Given that $f(\theta)$ is the uniform distribution between $\pi/20$ and $\pi/2$, as this is symmetric, then the expected error for a single voxel observation will be:

$$\begin{aligned}\mathbb{E}[r_p(\theta)] &= \int_{\frac{\pi}{20}}^{\frac{\pi}{2}} \frac{20}{9\pi}(d\sin(\theta) - d)\,\mathrm{d}\theta \\ &= -0.3014d\end{aligned} \tag{6.7}$$

Note that $d$ has an upper bound of the truncation distance $\delta$.

However, this does not consider multiple observations of the same voxel, which will lower this error. To quantify this, we performed Monte Carlo simulations of merging multiple independent measurements of the same voxel, shown in Fig. 6.3. The results show that even for as few as 3 observations, the error has an upper bound of $0.5\delta$ with $p = 0.95$, and as the number of observations increases, the error at this probability is reduced down to below $0.25\delta$.

Depending on how large the fixed band is determines how much to compensate for this error. If only a single voxel of the surface frontier is used, then it is safe to increase the safety distance by half of one voxel.

The second source of error considered is from the quasi-Euclidean distance assumption in the ESDF calculations. Quasi-Euclidean distance is measured along horizontal, vertical, and diagonal lines in the grid, leading to no error when the angle $\phi$ between the surface normal and the ray from the surface to the voxel is a multiple of 45°, and a maximum error at $\phi = 22.5°$ [70]. If $\phi$ is uniformly distributed between 0 and $\pi/4$ the residual $r_q(\phi)$ for this error, and its maximum and expected values are:

$$r_q(\phi) = \Big(d - \frac{d\sin(5\pi/8 - \phi)}{\sin(3\pi/8)}\Big) \tag{6.8}$$

$$r_q(\frac{\pi}{8}) = -0.0824d \tag{6.9}$$

$$\begin{aligned}\mathbb{E}[r_q(\phi)] &= \int_0^{\frac{\pi}{4}} \frac{4}{\pi}\Big(d - \frac{d\sin(5\pi/8 - \phi)}{\sin(3\pi/8)}\Big)\,\mathrm{d}\phi \\ &= -0.0548d\end{aligned} \tag{6.10}$$

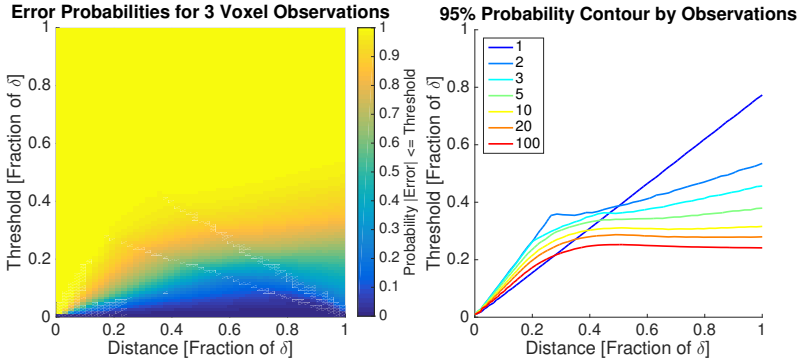



**Figure 6.3:** Probability of the distance error being below a threshold, for a given voxel distance measurement. a shows the probabilities for 3 voxel observations, and b shows 95% probability contours for multiple observations. For a single voxel observation, the maximum error with $p = 0.95$ is $0.8\delta$, while for 3 observations $p = 0.95$ falls at $0.5\delta$, and trends towards $0.25\delta$ as the number of observations grows.

Since the $d$ in this case only has an upper bound in the maximum ESDF computed distance, we recommend inflating the bounding box of the robot by 8.25%. Section 6.2 has empirical results on what effect this assumption has on the overall error in the ESDF computations, and shows that in practice it is small enough to justify the speed-up between full Euclidean and quasi-Euclidean distance.

# 6 Experimental Results

In this section we validate the algorithms presented above on two real-world datasets: the cow dataset with an RGB-D sensor and EuRoC with a stereo camera, both validated against structure ground truth.

The cow dataset[1] features several objects including a large fiberglass cow in a small room. It is taken with the original Microsoft Kinect, uses pose data from a Vicon motion capture system, and the ground truth is from a Leica TPS MS50 laser scanner with 3 scans merged together.

The EuRoC dataset is a public benchmark on 3D reconstruction accuracy [9], in a medium-sized room filled with objects. It is taken with a narrow-baseline grayscale stereo sensor, using Vicon fused with IMU as pose information, and

[1] projects.asl.ethz.ch/datasets/doku.php?id=iros2017

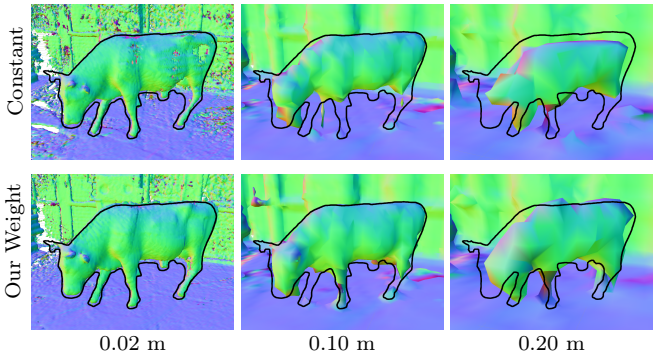



**Figure 6.4:** Qualitative comparisons of weighting/merging strategies on the cow dataset, colored by normals and with the object outline from ground truth overlaid. As can be seen, especially at large voxel sizes, our weighting strategy distorts the structure less.

Leica TPS MS50 scans as structure ground truth. We use the `V1_01_easy` dataset for experiments.

All experiments are done on a quad-core i7 CPU at 2.5 GHz. Only one thread is used.

## 6.1 TSDF Construction

In order to verify that our weighting strategy scales well with larger voxel sizes, we validate our TSDF reconstructions against the structure ground truth for our datasets.

We evaluate the accuracy of our reconstruction by projecting each point in the ground truth pointcloud into the TSDF, performing trilinear interpolation to get the best estimate of the distance at that point, and taking that distance as an error. We consider only known voxels, and allow a maximum error equal to the truncation distance ($\delta = 4v$).

Qualitative comparison are shown on the cow dataset in Fig. 6.4, compared to the ground truth cow silhouette. As can be seen, constant weighting significantly distorts the geometry of the cow at larger voxel sizes: the head is no longer in the correct position, and the rear legs are gone entirely, which will lead to incorrect distance estimates in the ESDF, while our weighting strategy better preserves structure.

Fig. 6.5 shows a quantitative comparison on both datasets with respect to voxel size: as can be seen, weighting has a more significant effect on error as voxel

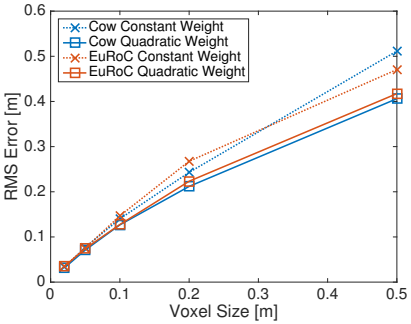



**Figure 6.5:** Structure accuracy reconstruction results for TSDFs with various voxel sizes, and comparing constant weight and quadratic weight with linear drop-off behing the surface. It can be seen that the choice of weighting function makes a more significant difference at larger voxel sizes, as more measurements are combined into any given voxel.

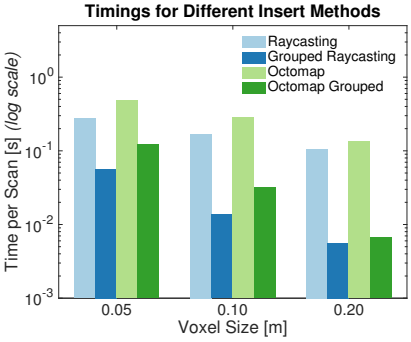



**Figure 6.6:** Timing results for different merging strategies on the EuRoC dataset. Our approach is up to 20 times faster than standard raycasting into a TSDF, and up to 2 times faster than even grouped Octomap insertions. Note log time scale.

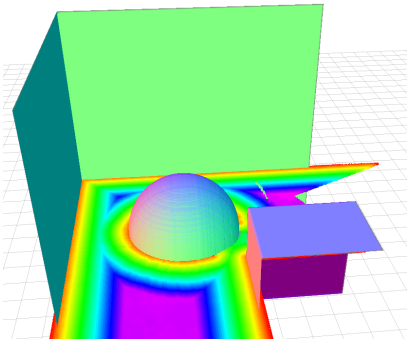


**Figure 6.7:** A normal-colored ground-truth mesh of the simulation experiment, with 3 planes (not pictured: ground plane), a cube, and a sphere. Also shown is a horizontal slice of the ESDF generated from 50 random viewpoints with a voxel size of 0.05 meters.

size increases, and our proposed quadratic weighting always outperforms constant weighting.

A comparison of the timings between various merging strategies and against Octomap [38] is shown in Fig. 6.6. While Octomap with the grouped raycasting strategy as discussed in Section 4.2 is already significantly faster than normal raycasting Octomap, it is still substantially slower than our TSDF approach. This is due to the hierarchical data structure: as the number of nodes in the Octomap grows larger, look-ups in the tree get slower, as they scale with $\mathcal{O}(\log n)$; with voxel hashing [76], the lookups remain $\mathcal{O}(1)$. Grouped raycasting leads to significant speeds up, especially with larger voxel sizes (as more points project into the same voxel). Overall, we show that using our merging strategy makes using TSDFs feasible on a single CPU core, allowing it to be used for real-time mapping and planning applications on-board an MAV.

## 6.2 ESDF Construction

### Simulation Results

To evaluate the errors introduced by various ESDF construction methods, we set up a simulated benchmark with 3 planes, a sphere, and a cube, shown with a horizontal ESDF slice in Fig. 6.7, of size $10 \times 10 \times 10$ meters. We simulated a noiseless RGB-D sensor with a resolution of $320 \times 240$ and a maximum distance of 5 meters. Readings were taken at 50 random free-space locations, uniformly sampled from all 6 DoF poses in the space that were a minimum of 1 meter from an obstacle.

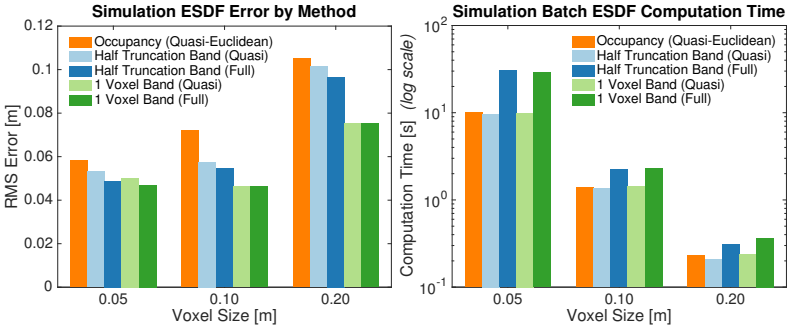



**Figure 6.8:** A comparison of methods for generating ESDFs from TSDFs and occupancy, and their errors and timing differences. Using all of the data in half the truncation distance is more accurate thanoccupancy, while using only one voxel-width of the surface outperforms all other methods, as shown in a. Quasi-Euclidean distance has only a marginal increase in error for a large decrease in computation time b.

We produced ground truth ESDFs of the space by evaluating the minimum distance to the objects at voxel centers. We then built a TSDF out of the simulated sensor data, and used multiple ESDF building methods to compare their error against this ground truth, as well as their integration times, shown in Fig. 6.8. The most basic method, occupancy, treats all negative-valued TSDF voxels as occupied and assigns them a distance of 0, similar to Wagner *et al.* [111] The next set of methods takes a band of values around the surface of the TSDF of half the size of the truncation distance ($\delta/2 < |d|$), and we compare both full Euclidean and quasi-Euclidean distances. The last set of methods takes a one-voxel-wide band around the surface, with both quasi-Euclidean and Euclidean distance.

As can be seen, the lowest errors are found by taking a one-voxel-wide fixed band around the surface. This is due to the projection error discussed in Section 5.2. However, it is important to note that all of the methods have a significantly lower error than using occupancy values, showing the advantages of building these maps out of TSDFs rather than occupancy maps.

While using full Euclidean distance shows improvement in ESDF error of 8.23%, 5.18%, and 4.72% in the half-truncation distance method for voxel sizes of 0.05, 0.10, and 0.20 meters, respectively, the integration times are increased by 201.0%, 61.3%, and 33.9%. Given that real-time execution is one of the core goals of this approach, our findings show that for many applications, using quasi-Euclidean distance is a good trade-off between error and runtime.

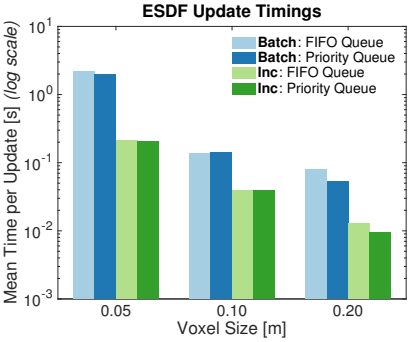



**Figure 6.9:** Timing results for updating ESDF in batch and incrementally, with different queueing strategies on the EuRoC dataset. The normal FIFO queue performs best for small voxel sizes, and at large voxel sizes, there is a speed-up from using a single-insert priority queue. Note the log time scale.

### Real Data

To validate the presented ESDF runtimes, we used real data from the EuRoC dataset for our evaluations on incremental and batch timings, using two different queueing methods, as discussed in Section 5. It can be seen in Fig. 6.9 that building the ESDF incrementally leads to an order of magnitude speedups over the entire dataset, and that at large voxel sizes, using a single-insert priority queue is faster than using a FIFO queue.

We also compare the integration time of the TSDF with update time of the ESDF layer in Fig. 6.10. Though for small voxel sizes, the ESDF update is slower than integrating new TSDF scans, at large enough voxels (here, $v = 0.20$ m), the TSDF integration time flattens out while the ESDF update time keeps decreasing. Since the number of points that need to be integrated into the TSDF does not vary with the voxel size, projecting the points into the voxel map dominates the timings for large voxels.

Based on these results, we recommend to use a single-voxel fixed band, quasi-Euclidean distance, and a priority queue for ESDF construction.

## 7 MAV Planning Experiments

To prove the usefulness of our generated ESDF for a real planning application, we set up an experiment with an MAV exploring an unknown space and replanning online as its ESDF gets updated. A photo and screenshot of this experiment is shown in Fig. 6.1, and the complete trial can be seen in the video attachment. The platform we used is an Asctec Firefly, equipped with a forward-facing stereo cam-

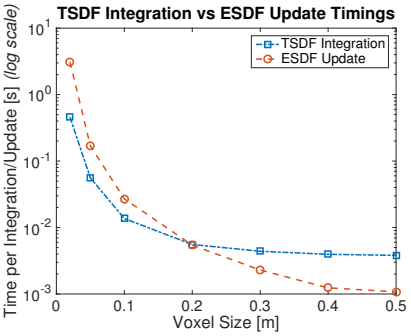



**Figure 6.10:** Timings results for integrating new data into the TSDF compared to propagating new TSDF updates to the ESDF on the EuRoC dataset. At small voxel sizes, TSDF integration is faster, but flattens out at large voxel sizes as the amount of sensor data does not decrease, while ESDF timings continue to decrease.

era synced to an IMU, which we use for stereo matching as input to the mapping process, and also as input to the visual-inertial state estimator. All state estimation, reconstruction, planning, and control runs entirely on-board on the Intel i7 2.1 GHz CPU without using any external sensing or infrastructure.

For this experiment, we use the continuous-time trajectory optimization replanning approach presented in [81], which relies on having smooth collision costs and gradients from a Euclidean distance map. We update the ESDF at 4 Hz, and replan after every map update, using 0.20 meter voxels. Even with random restarts in the optimization procedure, the complete system including TSDF construction, ESDF updates, and replanning was able to run well under the 250 ms time budget.

We made one extension to the planning method to guarantee good performance: since the planner can not handle unknown space (as there are no collision gradients available), we allocate a 5 meter sphere around the robot's current position and mark all unknown voxels in that sphere as occupied. To compensate for fact that the MAV can not observe its current position (and would therefore mark it as unknown), we take a smaller 1 meter sphere around the robot's start position and mark all unknown voxels in this smaller sphere as free. Note that these changes do not affect any voxels that have actually been *observed* – only *unknown* voxels are modified.

This experiment demonstrates that the proposed mapping approach can be used in combination with a planner and state estimator to navigate a small aerial robotic platform to a waypoint in a previously unknown environment while continually replanning to avoid obstacles. This is achieved while staying within the computational limits of the platform and operating in real time.

## 8 Conclusions

This paper aims to find a suitable map representation for local planning on MAVs in unexplored environments. Euclidean Signed Distance Fields (ESDFs) provide distance information to obstacles, which is essential for trajectory optimization planners. In contrast, Truncated Signed Distance Fields (TSDFs) are fast to build, filter out noise in sensor data, and can be used to easily create human-interpretable meshes. We propose to incrementally build ESDFs directly out of TSDFs, rather than occupancy-based representations. We extend existing methods to take advantage of distance information in the TSDF and allow dynamically-growing maps by using voxel hashing as the underlying data structure.

We focus on reducing the computational complexity of building these maps, while quantifying the errors introduced in our approximations to guarantee planning safety. Our results suggest building a TSDF with 20 cm voxels, using grouped raycasting, and quadratic weights with linear drop-off behind a surface boundary. We also recommend using a one-voxel fixed band from the TSDF in order to build the ESDF, using quasi-Euclidean distances, and a distance-based priority queue for processing the open set. Given possible sources of error in the maps, we recommend inflating the robot bounding box by $8.5\% + 0.3v$, where $v$ is the voxel size.

We show that our method of building the TSDF is faster than building an Octomap, and that the accuracy of an ESDF built from a TSDF is higher than if built from an occupancy map. Finally, we validate our complete system by building maps and using them to plan online with a trajectory optimization replanner, entirely on-board an MAV.

Paper IV

# Safe Local Exploration for Replanning in Cluttered Unknown Environments for Micro-Aerial Vehicles

Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto

**Abstract**

In order to enable Micro-Aerial Vehicles (MAVs) to assist in complex, unknown, unstructured environments, they must be able to navigate with guaranteed safety, even when faced with a cluttered environment they have no prior knowledge of. While trajectory optimization-based local planners have been shown to perform well in these cases, prior work either does not address how to deal with local minima in the optimization problem, or solves it by using an optimistic global planner.

We present a conservative trajectory optimization-based local planner, coupled with a local exploration strategy that selects intermediate goals. We perform extensive simulations to show that this system performs better than the standard approach of using an optimistic global planner, and also outperforms doing a single exploration step when the local planner is stuck. The method is validated through experiments in a variety of highly cluttered environments including a dense forest. These experiments show the complete system running in real time fully onboard an MAV, mapping and replanning at 4 Hz.

Published in:
*IEEE Robotics and Automation Letters (RA-L)*, 2018
DOI: 10.1109/LRA.2018.2800109

## 1 Introduction

Micro-Aerial Vehicles (MAVs) have the potential to perform many mapping and inspection missions for search and rescue and other humanitarian operations, where it is dangerous or impractical for humans to go. Planning is a key part of any autonomous system, and online local replanning allows for fast reactions to newly observed or dynamic parts of the environment. And while local replanning has also been recently addressed in literature, most work is shown on very low-density environments, and makes optimistic assumptions about the environment (for example, that unknown space can be treated as free before observing it) [14, 91].

However, in more cluttered, unknown environments, these assumptions may lead to poor planning results. Executing these plans can also be dangerous, both for the MAV and nearby people. For example, assuming unknown space is free in forest spaces can lead to planning directly upwards into the tree canopy, this can occur as obstacles directly above an MAV are often outside the field of view of its sensors. Alternatively if a highly conservative local planner is employed, many cluttered environment will result in the system finding no feasible paths to the goal. In this work, we present a system that combines a conservative local planner with a local exploration strategy to navigate a cluttered, unknown environment such as the forest in Fig. 7.1.

Different local optimization methods for avoidance have been recently covered in literature [19, 81, 109]. However, most do not explicitly address the problem of getting stuck in local minima. This poses a special problem in unexplored or partially unexplored environments, where only locally-optimal or reactive planners will frequently fail to find a path. Other approaches use an optimistic global planner (one that considers unknown space as free) to overcome the problem of occasionally getting stuck. While this works well in low-density environments, our work aims to show that this strategy (using an optmistic RRT* [44] for goal selection) is not effective for highly-cluttered, partially unexplored environments.

Instead, we bring in concepts from the exploration literature to the area of local replanning. We compare our optimistic global planning to performing an exploration step from the exploration-gain-based "next-best-view" planner (NBVP) when the trajectory optimization planner fails to find a feasible solution [3]. We then propose our own local exploration method, which tightly couples the local planning algorithm with a strategy that selects an intermediate goal. The method maximizes both coming closer to the final goal and potential exploration gain, increasing the chances of finding a feasible path.

To solve the problem of map representations, our method also uses an incrementally-built, dynamically-growing Euclidean Signed Distance Field (ESDF) to compute collision costs and gradients. The ESDF is built from a Truncated Signed Distance Field (TSDF) [83], and allows us to plan in initially unknown environments with no prior knowledge of upper bounds on map size, and does not require pre-computing the object distances in batch.

We compare different parameters for our underlying local optimization method, which is an extension of our previous work [81], when the map is known *a priori*

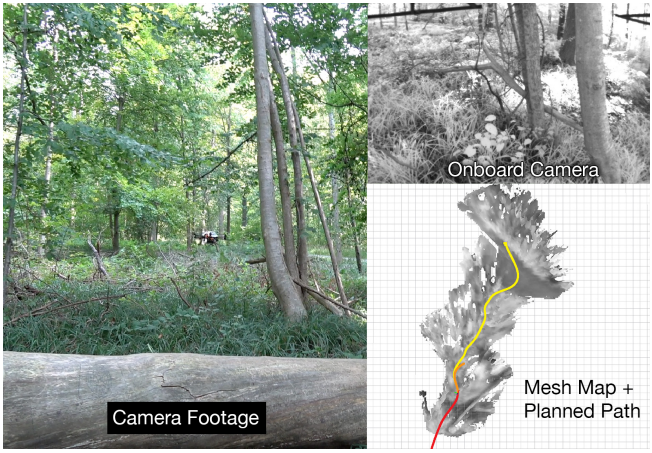



**Figure 7.1:** Experimental results from a flight through a dense forest, with video camera footage on the left, on-board view from one of the stereo cameras on the upper right, and a representation of the final mesh map on the bottom right. The final flown path is shown in yellow, the current pose of the MAV in the photos is shown as colored axes, and the planned path at the time of the photo is shown in orange.

or initially unknown, and then compare the success rates of various intermediate goal-finding strategies in highly cluttered environments. We then demonstrate our complete system running in real-time on-board an Asctec Firefly MAV and navigating without any prior map knowledge through both an office environment and a dense forest.

The contributions of this work are as follows:

- Extension of optimization problem for continuous-time polynomial trajectory optimization.
- A system, including mapping and planning, which conservatively handles unknown space and is able to grow the map over time.
- An active *local* exploration strategy for overcoming local minima even in unknown environments by finding intermediate goal points.
- Simulation benchmarks and real-world experiments in various cluttered environments.

## 2 Related Work

While a large number of methods exist for local avoidance, we will address methods in 3 categories. The first is purely reactive methods, which do not build a map of the environment but instead plan directly in the current sensor data. While these methods are very fast and computationally efficient, they do not work well in cluttered environments where avoidance maneuvers may be non-trivial, and suffer heavily from falling into local minima. The second class is map-based local avoidance methods, which use various techniques to compute feasible and locally-optimal paths through local maps built from sensor data or *a priori* known global maps. The last class of work we will examine here does not focus on obstacle avoidance, but instead on maximizing exploration coverage of unknown environments. While planning collision-free paths is also a requirement for any exploration strategy, the focus is on minimizing unknown space in the final map. We will draw inspiration from some of these methods to overcome the shortcomings of using optimization-based local planners alone.

### 2.1 Reactive Avoidance

Reactive methods focus on reacting to incoming sensor data as quickly as possible, and so act directly on obstacles in the current sensor field of view without building persistent maps.

For instance, our previous reactive work shows a method to directly convert incoming disparity maps from stereo into object segmentations, and then uses wall-following algorithm to avoid them [80]. Florence *et al.* directly integrates the nearest obstacle from a disparity map into a controller that is an open-loop library of motion primitives [22]. Only inexact, local state estimation is required for this approach, and they demonstrate it in both extensive simulation and real-world experiments. Lopez *et al.* build a kD tree of the current sensor view pointcloud, and then perform aggressive reactive avoidance from a library of fixed-velocity but variable angle motion primitives, generated from a triple-integrator model of MAV dynamics [60].

While all three methods are shown avoiding obstacles directly in front of the MAV without prior map knowledge, they are only demonstrated on much lower obstacle densities than discussed in this paper, and suffer from not being able to avoid obstacles that are not directly in the current sensor field of view.

### 2.2 Map-Based Replanning

In contrast, most replanning methods focus on navigating in a map rather than directly on sensor data.

Richter *et al.* presented dynamics-aware path planning for MAVs as solving an unconstrained QP through a visibility graph generated by an RRT [95], which remains a popular method for global planning [8], but is debatably too slow to replan in real-time. Our previous work [81] combines unconstrained polynomial spline op-

timization with gradient-based minimization of collision costs from CHOMP [93], but is prone to local minima. Usenko *et al.* utilize a similar concept, but use a B-spline representation instead, and use a circular buffer-based Octomap to overcome the issue of needing a fixed map size [109]. Dong *et al.* also use the same general problem structure as CHOMP, but represents trajectories as samples drawn from a Gaussian Process (GP) and optimize the trajectory using factor graphs and probabilistic inference [19]. While all these methods are able to avoid obstacles and replan in real time, none offer convincing ways to overcome the problem of getting stuck in a local minima and being unable to find a feasible solution.

Pivtoraiko *et al.* use graph search with motion primitives to replan online [91]. However, they use an optimistic local planner: unknown space is considered traversible, and while this helps escape local minima, it is fundamentally unsafe. Chen *et al.* plan online by building a sparse graph by inflating unoccupied corridors within an Octomap, then optimize an unconstrained QP to get a polynomial path [14]. However, they only use 2D sensing and treat unknown space as free, again leading to potentially unsafe paths in very cluttered environments.

## 2.3 Exploration

The goal of exploration literature is not only to stay safe and avoid collisions, but to maximize the amount of information about the environment. There are many different approaches, such as greedily tracking the closest unexplored frontier [34] or simulating gas-like particles throughout the environment to find the sparsest area of dispersion to explore [103].

Rather than tracking frontiers, some methods instead aim to maximize information gain. Charrow *et al.* optimize this gain over a state lattice with motion primitives as connecting edges, and then improve the plan with trajectory optimization [13]. Bircher *et al.* instead build an RRT tree in the unexplored space, and execute a straight-line plan to the first vertex of the most promising branch of the tree, maximizing the number of unknown voxels falling into the sensor frustum [3]. Papachristos *et al.* extend Bircher's method by also optimizing the intermediate paths to maximize localization quality [88]. Similarly, Davis *et al.* optimize paths between next-best views to maximize coverage by introducing a coverage term to their iLQG formulation [17].

Our work combines the fast online replanning capabilities of trajectory optimization-based planning with the idea of maximizing exploration gain in a future sensor field of view. This combination allows us to overcome the tendency of local planners to get stuck with local minima, while intelligently using our model of the system to find feasible solutions.

# 3 Problem Description

We aim to solve the problem of an MAV attempting to reach a goal in a previously unexplored (and completely unknown) environment. The core focus being on very

obstacle-dense and cluttered environments, with forest flight as a particular example. The MAV has at least one 3D imaging sensor, either RGB-D or stereo, with a finite resolution and a fixed horizontal and vertical FOV, mounted in a fixed position. We assume that the MAV is building a map of the environment from this sensor as it navigates (Section 5). We design a conservative local planner, which treats unknown space as occupied and inaccessible (Section 4). The core problem we want to address is how to design a complementary goal-finding algorithm for when the local planner gets 'stuck' in a local minimum (Section 6). All parts of the method should be fast enough to run online and in real-time entirely on-board the MAV.

## 4 Local Trajectory Optimization

Our local trajectory optimization method is an extension of our previous work [81]. We represent an MAV trajectory as a high-degree polynomial spline as in Richter *et al.* [95], and put soft constraints (expressed in the segment time allocation) on the maximum velocity and acceleration along the trajectory, which Mellinger *et al.* show makes the trajectory physically feasible for a simplified dynamics model [66].

The actual optimization minimizes a compound cost, consisting of minimizing a derivative of position such as jerk or snap as in [95] and [66], combined with the collision gradient cost from Ratliff *et al.* [93].

We will consider a polynomial trajectory in $K$ dimensions, with $S$ segments, and each segment of order $N$. Each segment has $K$ dimensions, each of which is described by an $N$th order polynomial:

$$f_k(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \dots a_N t^N \tag{7.1}$$

with the polynomial coefficients:

$$\mathbf{p}_k = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_N \end{bmatrix}^\top. \tag{7.2}$$

In order to avoid numerical issues with high orders of $t$, we instead optimize over the end-derivatives of segments within the spline [95], sorted into fixed derivatives $\mathbf{d}_F$ (such as end-constraints) and free derivatives $\mathbf{d}_P$ (such as intermediate spline connections):

$$\mathbf{p} = \mathbf{A}^{-1}\mathbf{M}\begin{bmatrix}\mathbf{d}_F \\ \mathbf{d}_P\end{bmatrix}. \tag{7.3}$$

Where $\mathbf{A}$ is a mapping matrix from polynomial coefficients to end-derivatives, and $\mathbf{M}$ is a reordering matrix to separate $\mathbf{d}_F$ and $\mathbf{d}_P$.

The final form of the optimization problem is:

$$\mathbf{d}_P^* = \underset{\mathbf{d}_P}{\operatorname{argmin}}\ w_d J_d + w_c J_c + w_g J_g \tag{7.4}$$

Where the derivative cost, $J_d$, aims to minimize a certain derivative (often jerk or snap) of the position [66], with $\mathbf{R}$ as the augmented cost matrix.

$$\begin{aligned} J_d &= \mathbf{d}_F^\top \mathbf{R}_{FF} \mathbf{d}_F + \mathbf{d}_F^\top \mathbf{R}_{FP} \mathbf{d}_P + \\ &\quad \mathbf{d}_P^\top \mathbf{R}_{PF} \mathbf{d}_F + \mathbf{d}_P^\top \mathbf{R}_{PP} \mathbf{d}_P \end{aligned} \tag{7.5}$$

The collision cost, $J_c$, is an approximation of the line integral of costs along the path, where $c(\mathbf{x})$ is the collision cost from the map, $\mathbf{f}(t)$ is the position along the trajectory at time $t$, and $\mathbf{v}(t)$ is the velocity at time $t$:

$$J_c = \sum_{t=0}^{t_m} c(\mathbf{f}(t)) \left\| \mathbf{v}(t) \right\| \Delta t \tag{7.6}$$

We use 3 segments and optimize jerk, as was found to be the best settings in our previous work [81].

We extend our previous work by using a soft cost for the goal, $J_g$, similarly to [109], and the local goal finding below.

$$J_g = \left\| \mathbf{f}(t_{\text{end}}) - \mathbf{g} \right\| \tag{7.7}$$

where

$$f_k(t_{\text{end}}) = \mathbf{T}_{\text{end}} \mathbf{A}^{-1} \mathbf{M} \begin{bmatrix} \mathbf{d}_{Fk} \\ \mathbf{d}_{Pk} \end{bmatrix} \tag{7.8}$$

This allows the optimization to slightly adjust the goal point to allow better trajectories, or find feasible trajectories at all. An analysis of the effect of this term on the success rate is offered in Section 7.

In general, even with the soft cost term, the initial state of the optimization problem should have the end point be free or almost free of collisions. In our system, we set a fixed planning horizon $r_p$, which is the maximum distance from the current state that the planner is allowed to go to. However, projecting a global goal $\mathbf{g}_g$ onto the sphere of this radius often leads to occluded end points.

In Section 7, we compare two different strategies for moving this end-goal to be a feasible end point for the spline: straight-line goal finding, which backtracks along the line from the projection of $\mathbf{g}_g$ to the start point of the trajectory, $\mathbf{x}_s$, until the first unoccupied point along this line. The second method is gradient-based in the map: from the projection of $\mathbf{g}_g$ onto the sphere, we evaluate the gradient of the collision cost map and follow the gradient down until a free-space location is found. If the approach becomes stuck in a local minimum of the gradient, we evaluate the straight-line strategy for one step.

Finally, these trajectories are only planned on $\mathbb{R}^3$ and derivatives. To map these trajectories to the full pose of the MAV on $SE^3$, pitch and roll are defined by the acceleration in $x$ and $y$ directions, while yaw $\gamma$ remains free. We use velocity-tracking yaw to increase the chances of the MAV seeing new or dynamic obstacles before collision:

$$\gamma(t) = \arctan\left(\frac{v_y(t)}{v_x(t)}\right) \tag{7.9}$$

## 5 Map Representation and Unknown Space

As the optimization method in Section 4 requires not only distances to the nearest obstacles but also the gradients of these distances, we require a map representation that can be efficiently queried for this information. While our original work [81] used a fixed-size Euclidean Signed Distance Field (ESDF) built from an octomap representation, we more recently presented a way to build ESDFs from Truncated Signed Distance Fields (TSDFs) efficiently. This allows the system to incrementally build maps of arbitrary size from sensor data in real time. This system, called *voxblox*[1], is used as the map representation for the proposed planner [83].

The map consists of both the original TSDF, built from sensor data, which contains projective signed distances to surfaces within a very small truncation distance to the object and free space information, and the ESDF which contains Euclidean distances to obstacles in a much larger radius. The details of how to build both representations incrementally is addressed in [83].

To implement the desired property of treating unknown space as occupied, we modify the ESDF with data from the current state of the robot. One critical issue with treating unknown space as occupied is that the starting position of the robot will never be observed and will always be treated as occupied. For this reason, we change the ESDF values of unknown voxels in a small clearing radius $r_c$ around the initial pose of the MAV to free. $r_c$ should ideally be only slightly larger than the collision checking radius of the robot.

We also take a large radius $r_o$, which should be greater than or equal to the maximum planning radius, and set all unknown voxels in this radius to occupied. Marking unknown as occupied is essential to conservative local planners, as allowing free entry into unknown space leads to behaviors such as slamming into the ceiling when presented with obstacles in front.

## 6 Intermediate Goal Selection

In addition to the mapping and local planning methods presented above, we need an active exploration strategy to overcome the shortcomings of local trajectory optimization methods in very cluttered, partially unknown environments. A typical solution to this problem is to use an optimistic global planner, which assumes unknown space is free, to select a new set of waypoints to track [77].

In the results section (Section 7), we quantitatively compare four core methods of selecting new waypoint locations. The first method is naive random waypoint selection. When the local optimization fails, we select a new 3D waypoint position at random within a sphere of the starting position of the trajectory. The planner then attempts to track this waypoint, until it is either reached or another infeasible solution is encountered. Then the new waypoint is set to the original goal point. This strategy (one random, one back to original goal) continues until either the original goal point is reached or the maximum number of replans is exceeded.

[1] github.com/ethz-asl/voxblox

The second strategy is an optimistic (unknown = free) RRT* [44] visibility graph. This aims to best simulate the global planners used in other approaches, such as [109] and [8] [77]. Since as Section 5 describes, we set a large radius of unknown space to occupied in the ESDF, we instead use the raw TSDF as the obstacle map and treat unknown voxels as unoccupied. We then generate a sparse visibility graph toward the final goal, and track the first waypoint in the graph. If the first waypoint is reached, then we keep iterating through the graph until the goal point. If at any time, the local planner is again stuck, we generate a new RRT* plan.

We also consider the opposite strategy; a conservative or pessimistic RRT*, which assumes unknown space is occupied. Since the underlying local planner is also conservative, if it is unable to find a solution, it is likely that no solution to the goal exists through free space. Therefore, we build the RRT graph and select the node in the tree that has the closest Euclidean distance to the goal point, and then track the first vertex in the branch of the tree that the closest node belongs to.

The next strategy we consider is directly from the exploration literature, the "next-best view" planner (NBVP) from Bircher *et al.* [3]. Their approach consists of building a rapidly-exploring random tree (RRT) with a small number of nodes in position and yaw space, and simulating the expected view fustrum of the camera sensor. The approach then selects the first node to execute in the branch that leads to the highest information gain in terms of unknown voxels that would be observed. We implement this approach for comparison; however, since there is no goal-tracking component to this exploration strategy, we use the same scheme as with the random waypoint selection: one exploration waypoint, followed by trying to reach the goal, followed by another exploration waypoint.

The final strategy is our exploration strategy, combining aspects of both the exploration strategy above and goal-tracking and sensor field of view awareness, described in detail below.

## 6.1 Proposed Method

Our method uses a similar methodology to NBVP, where the potential exploration gain of future points is evaluated by projecting the camera frustum into the voxel grid. However, we adapt the method to (i) better suit the purpose of increasing the chances of the robot making it to the goal, and (ii), to function online, in real-time in a high-rate loop. The core differences are that we do not build an RRT graph, we subsample within the view frustum, we do not do raycasting to find occlusions, and we introduce a goal-seeking reward in addition to the exploration gain.

Our method works as follows: first, we draw the global goal $\mathbf{g}$ with some probability $P_g \in (0, 1)$. Otherwise, we proceed to generate $N$ random points, $\mathbf{x}_n$, in the *unoccupied* space of the TSDF, within a maximum radius $r$ of the start point of the trajectory $\mathbf{x}_s$. Note, importantly, that we use the original TSDF rather than the ESDF to select these points and evaluate the frustum, as the ESDF sets nearby unknown space to occupied for planning safety purposes.

We select a yaw $\gamma$ for each point by finding the angle of the vector from the

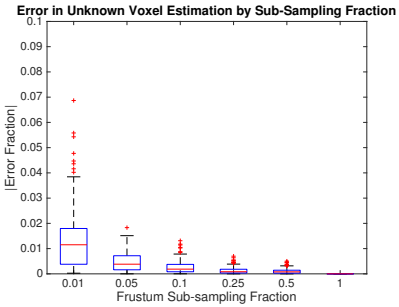



**Figure 7.2:** Error in estimation of unknown voxels in the sensor frustum (as a proxy for exploration gain), by subsampling fraction (a subsampling fraction of $0.01 = 1\%$ of the samples are taken). As can be seen, a sampling of 5% of the samples yields only a maximum 4% error in the unknown voxel estimation but could lead to up to a $20\times$ speedup in lookup operations.

trajectory start $\mathbf{x}_s$ to the sampled point $\mathbf{x}_n$, to approximate the real velocity-facing yaw. For each of these points, we evaluate the exploration gain of the camera frustum at that point by counting the number of unknown voxels in the TSDF. The exploration gain function $l(\mathbf{x}, \gamma)$ can be expressed as:

$$l(\mathbf{x}, \gamma) = \#\{v | v \in \text{frustum}(\mathbf{x}, \gamma) \cap v \in \text{unknown}(v)\} \tag{7.10}$$

In order to run in real-time, we approximate the actual exploration gain by subsampling the frustum by a certain factor, and checking only every $s$th voxel. We evaluate the effect of this approximation in Fig. 7.2 in simulation, which shows that sampling only 5% of the samples usually leads to an estimation error of less than 1%, and in practice runs 3 times faster than evaluating the full frustum.

Additionally, for each point we also evaluate the distance to the global goal, normalized by the maximum distance to goal $d_g$ (to allow consistent weighting across different settings and goal distances). This normalized distance is converted to a reward, giving the total reward function $R$ for each point $\mathbf{x}_n$ as:

$$d_g = \|\mathbf{g} - \mathbf{x}_s\| + r \tag{7.11}$$

$$R(\mathbf{x}_n, \gamma, \mathbf{g}) = w_e l(\mathbf{x}, \gamma) + w_g \frac{d_g - \|\mathbf{g} - \mathbf{x}_n\|}{d_g} \tag{7.12}$$

The point with the highest reward is chosen as the next intermediate goal.
A diagram showing the complete system (including mapping) is shown in Fig. 7.3.

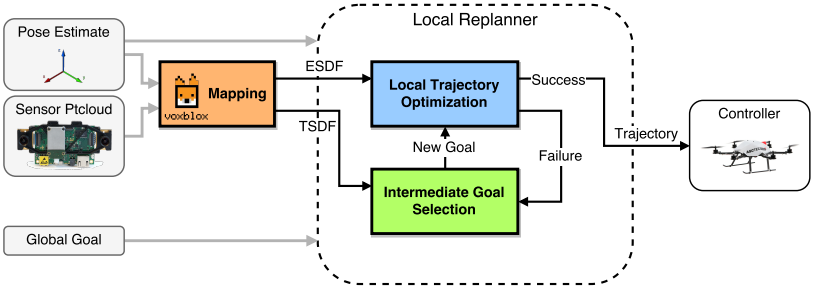



**Figure 7.3:** System diagram of the mapping and planning subsystems. The ESDF is used by the trajectory optimizer to compute collision costs, and the TSDF is used by the intermediate goal finding (local exploration algorithm) to evaluate exploration gain. If the trajectory optimization succeeds, the trajectory is sent to the controller; otherwise we attempt to find an alternative intermediate goal.

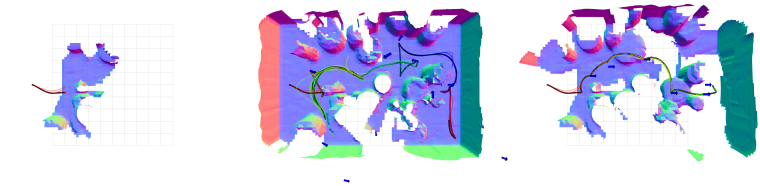
# 7 Simulation Experiments

This section will evaluate different aspects of our system in a simulation environment where the ground truth map is known. We compare the effects of full map knowledge vs. planning in an initially unknown map, evaluate the effect of parameters on success rate of local trajectory optimization, compare the intermediate goal finding methods presented in Section 6, and the effect of subsampling the camera view frustum for exploration gain evaluation.

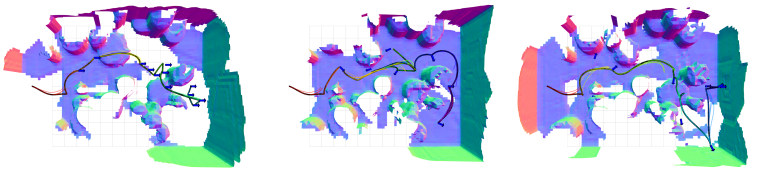
These simulations are made with the *voxblox*, which allows generating ground-truth ESDFs for environments made of primitive shapes (in this case, cylinders to simulate trees in a forest), and also allows simulating sensor measurements by raycasting into the map. The maps are 15 meters $\times$ 10 meters, and have an obstacle region of $10 \times 10$, to ensure that the start and end poses are always free. Cylinders of radii between 0.1 and 0.5 m and various heights are placed randomly within the space. The objects per square meter metric maps to approximately to percentage of the volume occupied, $\pm 5\%$ (for instance, 0.4 objects/$\text{m}^2$ is 35-45% occupied volume).

For the purposes of these experiments, we assume our MAV can track the polynomial trajectories perfectly, which [66] shows is possible as long as we respect maximum velocity and acceleration bounds while planning. We add a new viewpoint into the incrementally-built map and then replan once a second of simulation time. The incremental planning methods have a maximum planning horizon of 3 meters.

The first results are for starting with an a completely empty map, and inserting new viewpoints along the path at 1 Hz, with a sample solutions from no incremental goal-finding shown in Fig. 7.4a and the quantitative results in Fig. 7.5. As



**(a)** No Inc. Goal **(b)** Random Goals **(c)** RRT* (Opt.)



**(d)** RRT* (Cons.) **(e)** NBVP **(f)** Our Method

**Figure 7.4:** Comparison of methods in a small simulation case (15 × 10 meters) with 0.3 objects/$m^2$ obstacle density. The black line shows the final path, the colored lines show intermediate paths, and dark blue arrows show the intermediate goals selected by the algorithm. Only our method and NBVP were successfully able to solve the case; both RRT* methods were unable to see the final location as free as they do not consider sensor field-of-view in the planning, and the random goal selection had too few replans. All methods ran for up to 120 replans.

can be seen, straight-line goal finding and purely local optimization are able to solve only a very small percentage of the test cases. Using gradient-based goal finding significantly increases the performance, and soft goals further increase success rate, especially at lower densities. However, the success rates overall are still unacceptably low.

To overcome these issues, we benchmark the intermediate goal finding methods, described in Section 6, on the same simulation cases. Example qualitative results are shown for all methods in Fig. 7.4. The simulations show the differences between the methods: random goals fails to find the goal within the allocated time as the intermediate goals are too undirected, and the two RRT*-based methods fail since they do not consider the field-of-view of the sensor and are therefore never able to observe the goal point as clear.

Fig. 7.6 shows the quantitative results: as can be seen, all goal-finding methods outperform the naive optimization-only method. The optimistic RRT* performs the worst, as it tends to select the same infeasible path over and over again as unknown space is marked as traversable for this method. NBVP performs somewhat better, as it uses the sensor model to maximize exploring the small area. Conser-

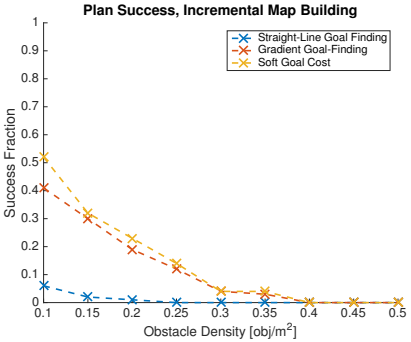



**Figure 7.5:** A comparison of the planner success without any intermediate goal-finding strategy, building the map incrementally. As can be seen, gradient-based goal-finding significantly increases success chance over line-based goal finding, and soft goal-cost further increases performance.There were 100 trials per density, with 60 replans (60 seconds at 1 Hz replanning rate).

vative RRT* performs comparatively well, as it is simply tracking the closest free point to the goal, but has no knowledge of the sensor model.

Finally, our method performs on par with random goal selection in terms of success rate. However, our method is able to consistently produce much shorter path lengths: Fig. 7.7 shows the mean path lengths for simulation cases that *both* random goal finding and our method were able to solve. Our method produces paths up to 35% shorter.

The final experiment is a more realistic test of a long forest traversal. We generate a 50 meter $\times$ 50 meter randomized map with 0.1 and 0.2 obstacles/m$^2$, and set the MAV to explore from one corner to the other. The results from 0.2 density are shown in Fig. 7.8, where our method and optimistic RRT* were the only two to successfully make it to the goal. Simulation results over 20 sims at different densities are shown in Fig. 7.9, and the timings of different aspects of our method from this simulation are shown in Table 7.1.

## 8 Real-World Experiments

To evaluate our system in a real-world scenario, we performed multiple experiments in two different test environments: a cluttered office space and a dense forest with a variable ground height. The results of all described experiments are available at `https://youtu.be/rAJwD2kr7c0`.

All of the experiments start with a completely unknown map, use visual-inertial odometry from the forward-facing (with a 12° downward pitch) stereo camera,

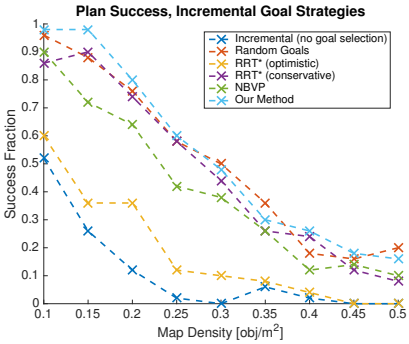



**Figure 7.6:** A comparison of the success rates for different incremental goal-finding strategies. Note that even though there is no prior map knowledge, these methods perform as well or better than pure replanning methods given full map knowledge. There are 50 trials per density, and a maximum of 120 replans per trial.

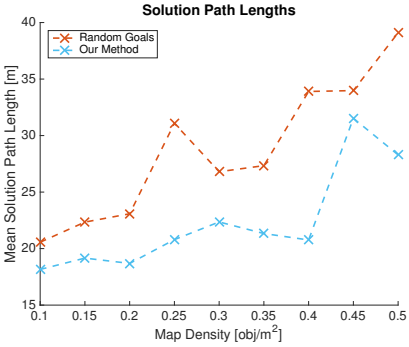



**Figure 7.7:** Path length comparison between random goal selection and our proposed method. The path lengths are only evaluated for trials where both planners succeeded, to allow a fair comparison. Note that our method always finds a solution in a significantly shorter path length, as it exploits current knowledge of the environment.

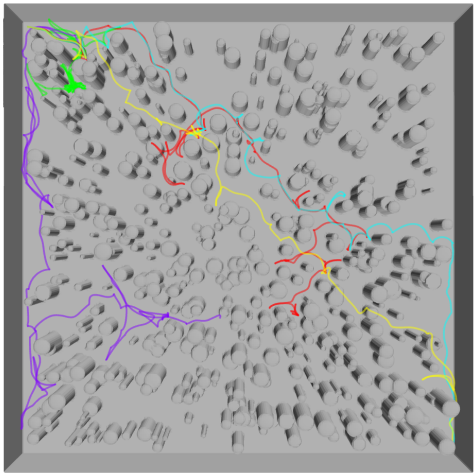


**Figure 7.8:** A 50 m × 50 m randomized “forest" environment, with a density of 0.2 objects/m$^2$. The lines compare five planners: no goal selection (dark blue), random goal finding (red), optimistic RRT* (yellow), conservative RRT* (purple), NBVP (green), and our method (teal). All planners start in the upper-left corner and try to reach the lower-right, but only our planner and optimistic RRT* are able to successfully find a solution in 500 replan cycles. The planning is in 3D, so some plans go over an obstacle.

update the map from stereo and replan at 4 Hz, and run everything entirely on the 2.4 GHz i7 dual-core CPU on-board the robot. We use rovio for state estimation [6], a non-linear MPC for position control [43], and the Asctec on-board attitude controller. The average flight velocity was 1.0 m/s.

In the office space environment, the MAV is able to navigate from a starting position in a hallway, around a corner, and to a point above an office table, shown in Fig. 7.10 During the path, it successfully avoids an ajar cabinet door (which blows open during the flight), along with many obstacles on either side of the hallway. The MAV was only able to reach near the intended goal, as it is unable to successfully determine whether the air-space above the tables is clear or not: the tables are gray and textureless, and the white projector screen behind them is also textureless, leading to a lack of stereo matches and therefore unknown space in the map. While eventually the robot would have explored enough of the surrounding space to clear this space, the pilot intervened when it was near the intended target. This demonstrates the conservative and safe nature of our planner.

Our second experimental validation took place in a forest environment, where we

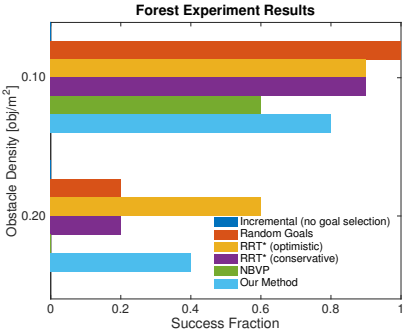



**Figure 7.9:** Quantitative results of success rate from the long forest simulation, limited to 500 replan cycles. While the RRT-based methods can offer good performance in this situation they also sample orders of magnitude more points than our method, and require much more time to select an intermediate goal.

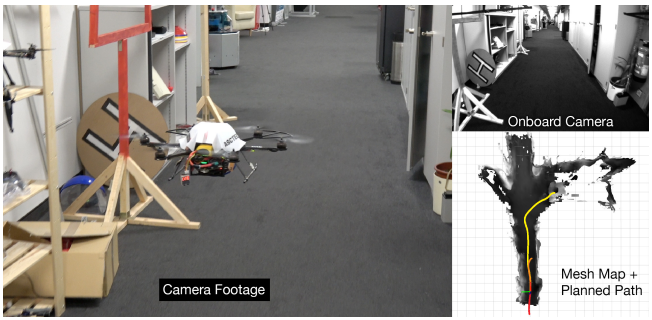



**Figure 7.10:** Experimental results from the office navigation experiment, with final map and intermediate paths shown on the lower right.

| Step | Time [$ms$] |
|---|---|
| **Mapping** | |
| TSDF Insert | 27.0 |
| ESDF Update | 14.5 |
| **Local Replanning** | |
| Trajectory Optimization | 19.3 |
| Intermediate Goal Selection | 5.9 |

**Table 7.1:** Timings for a single iteration each part of the method, aggregated from the benchmark in Fig. 7.8. Note that intermediate goal selection will only run if trajectory optimization fails, not every planning iteration.

performed four different experiments. In the first trial, we were successfully able to avoid a single large tree between the start point and goal. Second, we did two experiments where the MAV was commanded to go a large distance in its current facing direction, where the robot successfully avoided tree branches along its way and navigated largely along a hiking trail for up to 45.0 meters. In the shorter experiment, the MAV was able to reach its goal. In the longer one, it was unable to reach the final goal as the slope of the ground was too high and the tilted-down camera did not allow it to perceive enough open space to safely raise its flying height above the ground.

The final forest experiment tested navigation in very cluttered, obstacle-dense environments. The MAV was commanded to fly in a very densely-forested area between two trails, containing many small trees, branches, uneven terrain, and other obstacles. A still image of the video, along with the corresponding robots-eye view and the final executed path are shown in Fig. 7.1. The MAV was able to complete a path of 34.7 meters, successfully avoiding obstacles along the way, and finishing at the waypoint above the trail on the other side of the wooded region.

# 9 Conclusions

This paper presented a complete system for local obstacle avoidance, consisting of an underlying trajectory optimization method, which uses an Euclidean Signed Distance Field (ESDF) built by *voxblox* to get collision costs and gradients, coupled with an exploration-inspired intermediate goal finding strategy to escape local minima in the optimization. We showed that our combined method outperforms the common strategy of coupling an optimistic global planner with a conservative local planner. In the case of high obstacle densities, our exploration-based method is able to find solutions to more planning problems. We also outperform the next-best view exploration method for intermediate goal, as we are able to incorporate information about the global goal and reduce the runtime of the exploration gain

evaluation.

Our approach focuses on solving the case of very cluttered environments in previously unknown maps, and maximizing the chances of finding the goal while building the map. To demonstrate the performance of our method in real-world scenarios, we were able to successfully navigate through an office and through multiple forest environments while performing all processing in real-time on-board an MAV.

Paper V

# Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning

Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto

**Abstract**

Micro-Aerial Vehicles (MAVs) have the advantage of moving freely in 3D space. However, creating compact and sparse map representations that can be efficiently used for planning for such robots is still an open problem. In this paper, we take maps built from noisy sensor data and construct a sparse graph containing topological information that can be used for 3D planning. We use a Euclidean Signed Distance Field, extract a 3D Generalized Voronoi Diagram (GVD), and obtain a thin skeleton diagram representing the topological structure of the environment. We then convert this skeleton diagram into a sparse graph, which we show is resistant to noise and changes in resolution. We demonstrate global planning over this graph, and the orders of magnitude speed-up it offers over other common planning methods. We validate our planning algorithm in real maps built onboard an MAV, using RGB-D sensing.

Published in:
*IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018
DOI not yet available

## 1 Introduction

One of the most fundamental problems in robotics is planning paths through known maps, often referred to as global planning. While there are multiple classes of solutions to this problem, including sampling-based methods like RRTs [44] and search-based methods like A* [32], one class of solutions that has not been used for many real problems in 3D: topological planning.

Extracting the topology of a 2D map built by a ground robot to use for planning has been a well-studied topic, including methods that allow the topological graph to be created incrementally and maintained online [41, 52, 108]. Most of these methods start with a Euclidean Signed Distance Field (ESDF), also known as Euclidean Distance Transform (EDT), of the space, where each point in a 2D grid stores its distance to the nearest obstacle. From the ESDF, it is then possible to generate the Generalized Voronoi Diagram (GVD) of this space by finding all the points that are equidistant from two or more obstacles. This set of points represents the 'ridges' in the ESDF and is also known as the medial axis. With some additional filtering and generation rules, the GVD can be used to create sparse paths through the environment which maximize obstacle clearance and preserve topological connections within the space.

However, these approaches have scarcely been extended to 3D for robot planning. Hoff *et al.* [37] and Foskey *et al.* [24] explored using GVDs in 3D (generated by computing 2D slices of the GVD at various heights on GPU) given perfect CAD mesh data of the environment for path planning. To the authors' best knowledge, no work has generated and used 3D GVDs for path planning on noisy, real-world data.

The aim of this paper is to address this research gap, and explore methods to generate descriptive, topology-preserving sparse graphs of 3D space that are suitable for global path planning for Micro-Aerial Vehicles (MAVs). The main use-case we target is that of multi-session flying for industrial inspection or search and rescue applications. In our scenario, the MAV has an initial exploration mission to build a map of the space in which it will operate. This map is then refined and processed off-line, and a sparse topological graph representing the environment is created. This graph can then be queried for very fast initial global plans to return to any point in the previously-explored map, and can be further refined with polynomial optimization techniques presented in our previous work [81, 85] and in Fig. 8.1.

Our method starts from a dense voxel map of 3D ESDF values, built using *voxblox* [83], extracts the 3D GVD or medial axis using methods inspired by graphics skeletonization literature [25], and creates a skeleton diagram by preserving only 1-voxel-thick edges of the medial axis. We then fit a sparse graph, consisting of vertices and straight-line edges, to this underlying diagram, and ensure that it maintains connectivity. Both the map construction and planning methods are made available open-source as part of *voxblox*[1].

[1] github.com/ethz-asl/voxblox

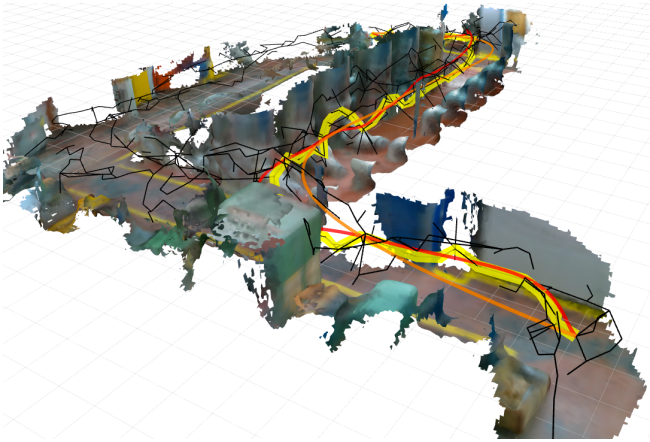


**Figure 8.1:** Sparse graph skeleton (black) and paths planned through a map created from on-board RGB-D sensing on an MAV in a machine hall. Yellow shows the initial path planned through the skeleton, orange is a dynamically-feasible path based on the initial path, and red is re-optimized to avoid collisions with the environment.

The contributions of this work are as follows:

- a novel method of extracting thin skeleton diagrams from real, noisy sensor data in dense voxel grids (Section 3),
- a method of constructing sparse, straight-line connected graphs to use for rapid planning, which is resistant to noise and resolution changes (Section 4),
- planning evaluations showing that our method can be orders of magnitude faster than other global planning methods (Section 6).

## 2 Related Work

In this section, we discuss existing robotics planning literature, with a focus on methods that exploit the topology of the scene. We split our discussion into 2D methods and 3D methods.

### 2.1 2D Planning

Thrun was one of the first to show topological grid-based planning in 2D for ground robots [108], where he built a topological map from Voronoi Graph, focusing on “critical points" (doorways, etc.) to divide into topologically separate regions, and

was able to show a three orders of magnitude speed-up over grid-based planning.

More recent work has focused on building signed distance fields and GVDs incrementally and in real-time. Kalra *et al.* introduced the original dynamic brushfire algorithm, which is able to maintain the accuracy of the underlying signed distance field by using raise and lower wavefront propagation, and by keeping an updated list of GVD candidates [41]. Lau *et al.* further extended this method to create one-voxel-thin GVDs that better represented the topology of the underlying space [52].

Fang *et al.* use 2D GVDs to inform sampling in 3D space for MAV navigation in indoor environments [21]. They build a 2D GVD from a down-projection of the environment, and use this to overcome the difficulties that sampling-based methods have with narrow corridors and openings. In contrast, our method actually builds a 3D GVD, fully capturing the geometry of the space.

## 2.2 3D Planning

We will cover three different categories of 3D planning: GVD-based 3D planning in CAD meshes, building topologically-aware maps based on something other than the GVD, and finally methods that explicitly model polygonal bounds of free-space regions for planning.

Hoff *et al.* presented a 2D but 3D-applicable method for building GVDs and then planning using potential fields in them. 3D is done with multiple 2D slices, and computed using graphics hardware [37]. The obstacles are from a labelled CAD model, and obstacles are defined per-object for the purposes of Voronoi boundaries (that is, a point belongs on the GVD if it is equidistant to two distinct *objects*, rather than any surface boundaries). Foskey *et al.* extended [37] to use a GVD for path planning directly in the graph, and if no solution through the GVD is found, use Probabilistic Roadmaps with sampling informed by the GVD.

Other methods for building topologically-aware graphs include SPARTAN, where a distance field is computed up to a certain distance from obstacles [15], and connected in straight lines between distance field boundaries. This representation is well-suited for very sparse environments, as it allows shortcuts through large open-space regions, but does bug-algorithm-style planning around obstacles. While their method is based on visibility graphs, we use Voronoi diagrams as our basis.

Another method is presented by Blochliger *et al.*, where SLAM-map landmarks projected into an occupancy map and then free-space is grown from the original robot map-inspection trajectory into mostly convex 3D clusters [5]. These clusters are then connected based on overlap to create a sparse navigation graph. The downside of this approach is that it heavily depends on the initial inspection trajectory rather than the inherent structure of the space.

Finally, another representation that has advantages for dynamics-aware planning is representing the environment as convex free-space regions. A well-known method is IRIS [18], which iteratively breaks up a world into polygonal free-space regions from a seed point, but takes hours to compute on any non-trivial 3D environment. Liu *et al.* attempt to overcome this speed limitation by building convex free space

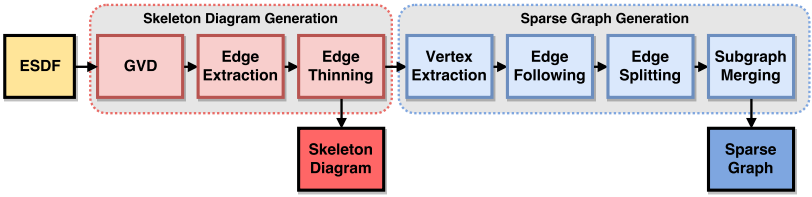



**Figure 8.2:** System diagram of the steps to generate the skeleton diagram (which is a dense voxel grid), and from there the sparse graph (which is an undirected graph with straight-line edges).

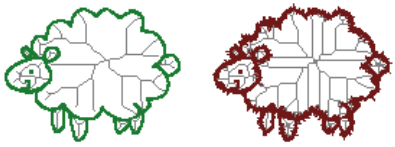


**Figure 8.3:** The effect of noise on the Voronoi diagram. The left image, uncorrupted by noise, has a very sparse diagram. The right image, generated with noisy sensor measurements, has many more edges in the diagram. Part of this work focuses on pruning the Voronoi diagram in the presence of heavy sensor noise in the map.

regions online, around an initial path found through A* or JPS graph search in a discretized space [59]. However, the final path will be a homotope of the initial A* path, so the free space regions can only be used for local refinement of the trajectory. Similarly, Chen *et al.* exploit the structure of their obstacle map representation, Octomap, to grow and merge free-space axis-aligned cubes around an initial path and find solutions through this reduced graph, and then refine them using polynomial trajectory optimization [14].

In contrast to all these methods, we compute the GVD-based topology of an entire noisy map, can be built on-line on-board an MAV, efficiently represents the underlying topology of a space, makes no assumptions about obstacle densities or using initial graph-search solutions.

## 3 Skeleton Diagram Construction

Our method consists of two main components: first, building a one-voxel-thick *skeleton diagram* in the voxel space, which preserves the topology and connectivity

of the original space while having as few elements as possible, as can be seen in 2D in Fig. 8.3. This is the closest analogue to the 2D topological maps used by Lau *et al.* [52]. Second, generating a *sparse graph* out of the diagram, which is no longer bound to the voxel space, and instead consists of graph vertices connected by straight-line edges.

The overall system diagram is shown in Fig. 8.2. The general order of steps is to generate the Generalized Voronoi Diagram (GVD), or medial axis, of the space, extract its edges and vertices, thin the diagram, then create a sparse graph by following edges in this diagram, and reconnect disconnected subgraphs in the final graph.

## 3.1 GVD Generation

In order to generate the original Generalized Voronoi Diagram, we follow the common approach of finding "ridges" in the signed distance function [24, 105]. Finding the GVD is analogous to finding a discretization of the medial axis of the free-space, which has been well-studied in 2D robotic planning problems [52, 108]. In 2D, the medial axis consists of connected lines that maximize the distance from obstacles, as shown in Fig. 8.3. However, since the medial axis has one dimension less than the original shape, when this is generalized to 3D, the medial axis consists of (potentially curved) planes that maximize this distance. To make the problem as sparse as possible for planning, we discard the planar representation and aim instead to find the medial curve skeleton, which consists of 1-voxel-thick lines in 3D. This is analogous to the edges and vertices of the 3D GVD, discarding the faces.

We begin with the same general approach as Foskey *et al.* [24] to find all the points that belong on the medial axis. To do this, we iterate over all voxels in the ESDF that are in free space, and compare their parent direction with the parent direction of their 6-connectivity neighbors. Each voxel stores the direction to its "parent" point on the object surface (the closest occupied point to the voxel). Note that we could also follow the 2D approach of Lau *et al.* [52] and store the GVD candidates from the termination of the wavefront to avoid having to check all the voxels in the map.

When the medial axis is constructed from CAD-generated data, where it is known which surface belongs to which object, it is sufficient to check if the neighbors have different parents or basis objects [37]. However, with noisy real-world data and discretized map representations, as show in Fig. 8.3, it is difficult to tell which points belong to which objects, and noise on the surface boundary creates many spurious medial lines.

To combat this problem, we adapt the $\theta$-SMA approach [25], which creates a simplified medial axis (SMA) based on a minimum $\theta$ angle between basis points. In our discretized ESDF, we compute whether a point belongs on the medial axis using:

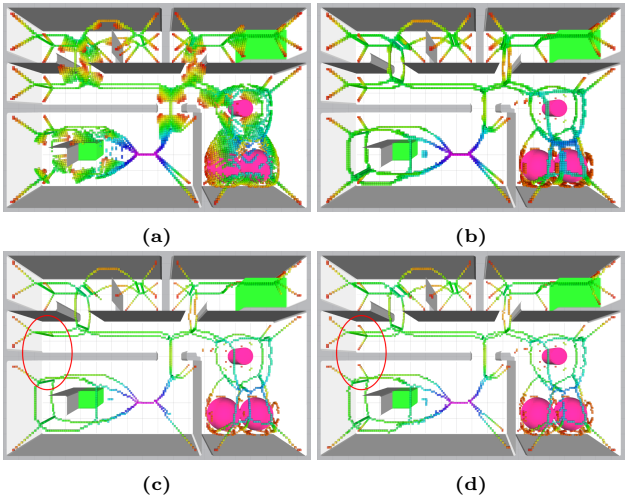



**Figure 8.4:** Different methods of generating the edges of the medial axis/GVD. a shows the edges generated by counting the number of basis points, b shows edges generated by number of neighbors that are also on the medial axis, c shows edges from b thinned using topology-preserving erosion techniques, and finally d shows the same technique applied but without extended end-point checks, note especially the edges preserved in the red circle. Color represents the distance to the obstacle.

$$\frac{\mathbf{n}_p + \mathbf{n}_d}{\|\mathbf{n}_p + \mathbf{n}_d\|} \bullet \frac{\mathbf{v}_p}{\|\mathbf{v}_p\|} < \cos\theta, \tag{8.1}$$

where $\mathbf{v}_p$ is the parent direction of the current voxel, $\mathbf{n}_d$ is the direction from the neighbor voxel to the current voxel, and $\mathbf{n}_p$ is the parent direction of the neighbor voxel.

Passing this check means that the point has at least two *distinct* basis points on object boundaries.

## 3.2 Edge Extraction

Now that we have all the points that belong on the medial axis, we need to classify them as edges, vertices, and faces, as we aim to only keep the one-voxel-thin lines that sparsely describe the topology of the space.

In a perfectly modeled environment, the edges of a 3D GVD are defined as having three basis points, and vertices as having four [37]. However, due to dis-

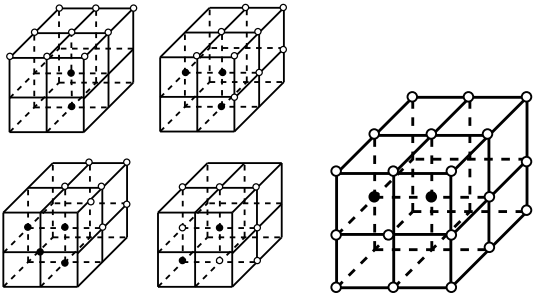



**Figure 8.5:** Voxel templates for thinning, a is the 4 deletion templates from [102], and b is our template to check whether a point is a 6-connected corner or not. Black points indicate foreground points (in our case, GVD edge points), white is background (non-edge points), and unlabelled match both foreground and background points.

cretization error, and the actual Voronoi boundaries falling between voxels, some edges disappear and spurious edges appear by this definition, as shown in Fig. 8.4a.

Since the Voronoi boundaries are also edges of the faces of the medial surface, we instead choose a method more stable to discretization error and noise in the boundaries: extracting edges of the medial axis, to create medial lines. We use the metric of how many of the voxel's 26-connected neighbors belong to the medial axis to determine whether it is an edge. We preserve all voxels with at least 18 neighbors, which creates the thick but complete skeleton in Fig. 8.4b.

However, even this definition is not free of errors introduced by discretization, especially depending on the size of the voxels. Lau *et al.* suffered the same problem in their 2D GVD, where many lines are two-voxels-thick, and create spurious connections. Their solution was to introduce two connectivity templates, which all pixels on the GVD were verified against to ensure that they were necessary to maintain the topology of the graph. However, since 3D topology is much more complex, it is not sufficient to verify our edge voxels against connectivity templates; we must instead do a topology- and connectivity-preserving 3D thinning operation, described in the section below.

## 3.3 Thinning

In order to remove spurious double lines, we refer to digital topology literature which builds 3D skeletons out of discretized shapes using recursive thinning [55, 63, 102].

We follow the general approach described in Lee *et al.*, which would also allow

parallelization of the thinning operation [55]. First, all voxels that are part of the shape are checked against one of several deletion templates. We choose to use the patterns from She *et al.* [102], due to their simplicity and not requiring more than the $3 \times 3$ neighborhood for every voxel, shown in Fig. 8.5a. The neighborhood of the center voxel is evaluated against these templates. In the templates, a point may be either foreground (black, edge or vertex neighbor), background (white, face or non-GVD neighbor), or "don't care" (unmarked). "Don't care" points can match against either value.

Once a point is shown to match one of the templates (or any of its 90° rotations or mirror operations), it can be removed if it is a *simple* point: that is, if the connectivity of its neighbors would be preserved without it [55], and not an end-point: has more than one 26-connected neighbor. This definition is both for groups of connected foreground voxels (connected using 26-connectivity) and connected background voxels (connected using 6-connectivity). Therefore, to verify that a point is simple, we need to verify that the number of connected components in its neighborhood remains the same without it. We do this using the octree adjacency tests proposed in Lee *et al.* [55].

However, since our edge-extraction heuristic produces 6-connected edges, removing all simple points that are not end-points leads to incorrect removal of some diagonal edges. This is due to the simplicity of the end-point test: a 6-connected end-point actually has *2* 26-connected edges. Classifying any point that only has one 6-connected neighbor leads to incorrect preservation of many other points. (This is not a problem encountered in medial skeleton construction through thinning, since the thinning is applied recursively and should always lead to only 26-connected components.)

To combat this issue, we extend the end-point test with a "corner template", shown in Fig. 8.5b. A point is considered an end-point if it has only one 26-connected neighbor, or if it has no more than one 6-connected neighbor and matches the template (and its rotations and mirrors) in Fig. 8.5b.

The results of this thinning operation is a fully-connected, one-voxel-thin skeleton, shown in Fig. 8.4d. This state of the map is referred to as the *skeleton diagram* for the remainder of the paper, and is analogous to the GVDs used for planning in 2D literature.

# 4 Sparse Graph Generation

To further sparsify the problem, we approximate the skeleton diagram with a sparse graph, using the steps described in this section.

## 4.1 Vertex Extraction

After our skeleton is only one-voxel-thick, we are able to extract vertices very simply: finding all edge voxels that have 1 or more than 3 26-connected neighbors that are edges. The results of this are shown in Fig. 8.6a, with red circles indicating

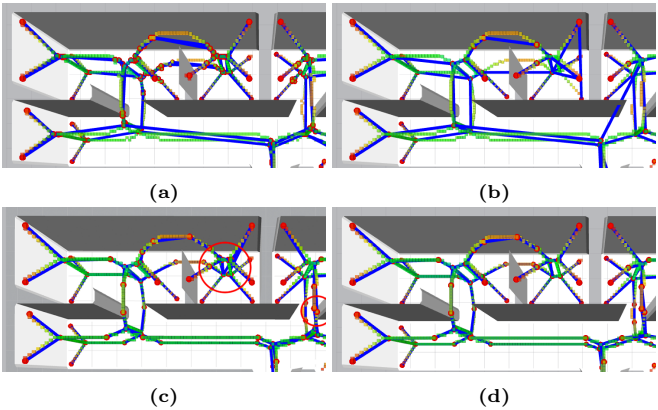



**Figure 8.6:** Steps in the process of vertex generation, with red circles as sparse graph vertices and blue connecting lines as sparse graph edges, overlaid on top of the GVD edges. a shows a connected sparse graph, without any pruning of vertices. b shows the result with k-D tree pruning; as can be seen, some edges pass through obstacles. c shows the results of splitting the edges any time they deviate too far from the straight line by inserting a new vertex, and d shows the results if the splitting also attempts to match to a nearby vertex (removing duplicated near-by vertices seen in c).

vertices. As can be seen, there are many redundant nearby vertices due to this definition. While they are technically correct, they clutter the sparse graph and add no new topologically-distinct paths.

In order to ensure that the graph we build in the next stage is as sparse as possible, we prune the vertices in the GVD. We build a k-D tree of the nearest vertices, and for each vertex in the GVD, find all other vertices that are within a pruning radius $r_{\text{prune}}$. From these vertices, the one with the largest distance to obstacles is retained, and all others are removed. The results of this operation are shown in Fig. 8.6b.

## 4.2 Edge Following

The next step is to trace straight-line edges between the sparsified vertices obtained from the previous section. The goal is to create a graph that is independent of the resolution of the underlying map, and is significantly faster to search through as all connectivity information is pre-computed.

We start with the filtered diagram vertices from the previous step. Each vertex is assigned a vertex ID, which is also marked in the diagram, and inserted into a

map of vertex IDs to vertex information. For each vertex, we attempt to follow the diagram edges to find connections to other vertices.

This is done as follows: first, for each vertex, we check its 26-connectivity neighbors for edges. For each edge, we recursively follow it through the diagram, by checking all its neighbors and preferring those most in line with the current direction of the edge:

$$\min\left((\mathbf{v}_d - \mathbf{r}_d) \bullet -\mathbf{n}_d\right), \tag{8.2}$$

where $\mathbf{v}_d$ is the direction taken to get to the current voxel, $\mathbf{r}_d$ is the direction from the vertex, and $\mathbf{n}_d$ is the direction from the current voxel to the checked neighbor.

Directions that are not followed are stored in a stack (FILO), and in case there are no more viable adjacent neighbors, the latest candidate is popped off the stack.

This procedure is followed until a vertex voxel is reached, then its corresponding entry in the sparse graph is found, and an edge connecting the originating voxel and the new one is created.

The results of the complete graph generation procedure, including edges shown as straight blue lines, is shown in Fig. 8.6b. As can be seen, some edges do not sufficiently represent the structure of the underlying diagram: for instance, some curved edges are represented as straight lines and as a result pass directly through an obstacle. This is obviously undesirable in a graph used for planning.

## 4.3 Edge Splitting

In order to prevent edges going through non-free space, we split edges that deviate too far from the straight-line path. This is done as follows: first, for every edge in the graph, we use diagram A* (described below in Section 5.2) to find the shortest path in the diagram from the start vertex to the end vertex. Every point in this diagram edge is then projected onto the straight-line between the start and the end vertex, and its distance to the line is calculated. If the maximum distance along this edge exceeds a threshold (for instance, we used twice the voxel size), then a new candidate vertex is created at the point on the diagram with maximum distance.

Making every candidate vertex into a real vertex gives the results in Fig. 8.6c: while the edges now match the shape of the diagram edges, there are some cases where unnecessary vertices are created: that is, locations where there is already a vertex nearby that the edge should have been connected to instead. To counteract this case, we add another check before adding a candidate vertex as a new vertex: if there is another vertex within $r_{\text{prune}}$ of the candidate, then use diagram A* to verify that a valid connection exists to both the start and the end vertices. This vertex candidate is considered *only* if connecting to it reduces the maximum distance to the straight-line.

The results of this final step are shown in Fig. 8.6d, where it can be seen that significantly fewer extra vertices are made, and shorter edges to existing vertices are established.

### 4.4 Disconnected Sub-Graph Repair

In some cases for large, cluttered maps, such as the maze we will use in Section 6.2, some edges that are present in the diagram are not re-connected correctly in the sparse graph due to discretization error. This leads to multiple disconnected sub-graphs.

We solve this problem by first determining the number of subgraphs and which vertices belong to them. For each vertex in the graph, if it is unlabelled with a subgraph ID, we assign it a new subgraph ID and perform the flood fill algorithm to label all the vertices that belong to its subgraph. The flood fill algorithm is a simple recursive depth first search, where we set the label of each vertex, follow all the edges from that vertex, and continue until no un-labelled vertices that are reachable by edges remain.

After each vertex is labelled, we perform two steps: first, remove all subgraphs that contain only one vertex, as these are not helpful in planning. Second, attempt to connect all the remaining subgraphs to each other.

We do this by using A* through the skeleton diagram. We select the first labelled vertex in two disconnected sub-graphs, and attempt to find a path between them along the underlying skeleton diagram. If A* is able to find a path, we trace back through it, checking every voxel along this path. For every voxel that is assigned a sparse-graph vertex ID, we record its ID and label. When the labels between two consecutive vertices change, we add an edge between those vertices and perform flood fill to re-label the new connected graph.

Note that this approach may connect more than just the start and end subgraph, if other disconnected subgraphs are encountered on the path. All new added edges are again checked for straight-line connectivity with the method described in the previous section.

We refer to the final state of this graph as the *sparse graph* for the purposes of planning.

## 5 Planning Algorithms

In this section, we will describe various ways to use the information contained in the ESDF, skeleton diagram, and sparse graph for path planning. These methods will be evaluated and compared in the results in Section 6.2. For each of these methods, we model the MAV as a sphere with a fixed radius. The GVD was generated with a minimum distance of this radius, so only valid traversible nodes are present in the graph.

For the first four planning methods, we focus on the goal of quickly finding a feasible initial path through the space, considering non-collision with obstacles as the only requirement. This initial path will then be refined to be a smooth, dynamically-feasible trajectory for the MAV using polynomial splines and the property of differential flatness, as described in Section 5.5 below.

## 5.1 A* through ESDF

The simplest method to find a path through the space is to run A* [32] on the ESDF voxels. For each voxel, starting from the start location, we expand its 26-connected neighbors and consider them part of the graph if their ESDF distance is greater than the robot radius. We use the straight-line distance to goal as an admissible heuristic, and accumulate voxel adjacency distances.

Though this approach is resolution-complete, it becomes prohibitively expensive in larger spaces or smaller voxel sizes due to the massive size of the graph.

## 5.2 A* through Skeleton Diagram

A much faster approach is to search only through the skeleton diagram. The heuristics and costs remain the same as in the ESDF A*, but a voxel is considered a valid neighbor if it is an edge or a vertex in the skeleton diagram.

One important consideration is that the start and end points may not necessarily be on the diagram. To counter this, we start an A* search through the ESDF from the start toward the goal, and abort as soon as it expands a vertex that is also on the diagram. We also start the same search backwards: from the goal toward the start. These two searches quickly find the start and end points on the diagram.

## 5.3 A* through Sparse Graph

The final speed-up is to traverse the sparse graph rather than the underlying diagram, as the graph keeps a mostly consistent size regardless of voxel size or noise level, as shown in Section 6.1.

We find the sparse graph vertices closest to the start and end points by searching for the nearest neighbors in a pre-built k-D tree structure. We then use A* to traverse the edges toward the goal vertex.

## 5.4 RRT through ESDF

We also compare to a more traditional method of global planning, based on sampling-based Rapidly-Exploring Random Trees (RRTs). We evaluate both RRT* [44], which is probabilistically-optimal (that is, it is guaranteed to find the optimal solution given infinite execution time) and RRT Connect [49], which has no optimality guarantees but in practice quickly finds a sub-optimal feasible path. In both cases, we treat unknown space as occupied and do collision-checking directly in the ESDF.

Unlike the search-based methods described above, which exhaustively search a graph toward the solution, RRT-based methods sample feasible points in the planning space and attempt to connect them to the existing tree. Once the goal state has been sampled and connected to the tree, the tree is traversed to get the final path. In RRT Connect, the tree is grown bi-directionally from both the start and goal. In RRT*, the initial path continues to be refined with shorter path-length solutions until a time limit is reached.

### 5.5 Dynamically-Feasible Trajectory Generation

Finally, given an initial straight-line path from any of these methods, we need to create a trajectory that the MAV can dynamically follow. We exploit the property of differential flatness to plan dynamically feasible polynomial splines. Since the splines may deviate from the initial straight-line solutions and no longer be collision-free, we use local trajectory optimization using the ESDF distances as collision costs to iteratively "push" the final trajectory out of collision, as described in our previous work [81, 85].

## 6 Experiments

In this section aims to validate our method on simulated and real-world data and analyze the effect of noise and voxel size on the resulting sparse graph. We also compare the different presented planning approaches, and finally show results on real maps from an MAV flight with generation of dynamically-feasible paths.

### 6.1 Sparse Graph Construction

We aim to show that the sparse graph construction method preserves the underlying structure of the scene, even in the presence of noise or different levels of discretization error (different voxel sizes). To do so, we construct a simulation world for which we have both the ground truth ESDF and are able to simulate synthetic viewpoints within this space, optionally corrupted by noise.

We generate the skeleton diagram and sparse graph for a variety of voxel sizes and noise magnitudes, shown in Fig. 8.7. The top row shows the sparse graph built from perfect ground truth data acquired from the simulation. The following nine scenarios show the same simulated world, reconstructed from 200 randomly selected robot poses with a simulated depth sensor. The sensor has a 90° field of view and $320 \times 240$ resolution. Optionally, we apply independent Gaussian noise to each distance measurement, with $\sigma = 0.1$ or 0.2. These scans are then incrementally integrated into a Truncated Signed Distance Field (TSDF), and we construct the ESDF from this TSDF using full Euclidean distances, as described in our previous work [83].

While it can be seen that noise and discretization error corrupt the graph by adding extraneous edges, the fundamental structure of the graph remains the same.

Another important effect of the sparse graph representation is that its size stays largely constant regardless of the size of the underlying diagram. This is shown in Fig. 8.8, where we compare the number of elements on the skeleton diagram (circles) and in the sparse graph (crosses). Even though the voxel size (and to a lesser extent, noise level) has a large impact on the number of elements in the skeleton diagram, the number of elements in the graph remains almost constant. This shows the robustness of our approach to different resolutions and corruption by noise.

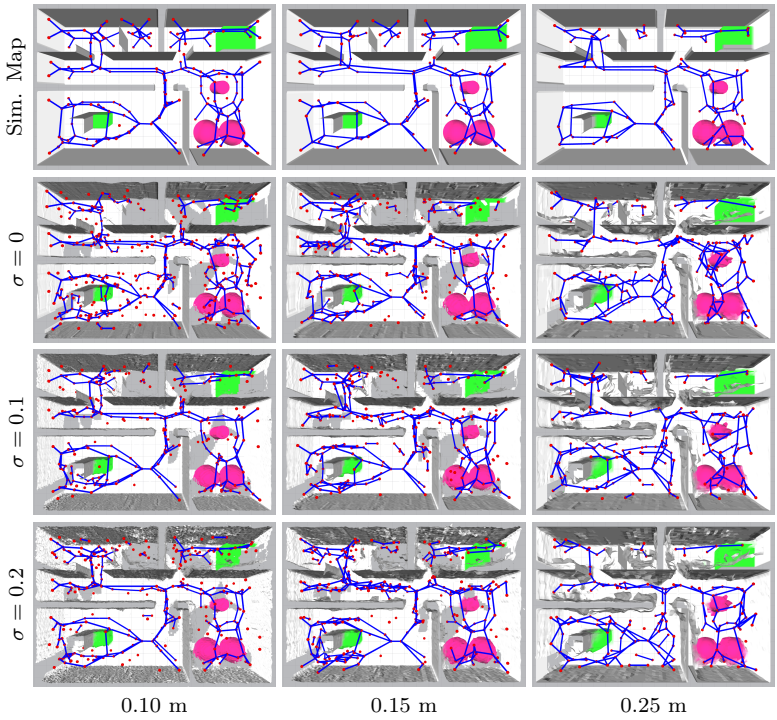



**Figure 8.7:** A comparison of sparse graphs generated from the same simulated environment, under different voxel sizes and amounts of noise in the distance measurements. The top row is simulated from ground truth ESDF data, while the other 3 rows show maps created from 200 simulated robot poses with a noisy depth sensor. As can be seen, despite extraneous edges introduced by noise or discretization error, the core structure of the graph remains the same. The diagram was generated with a minimum distance of 0.4 meters.

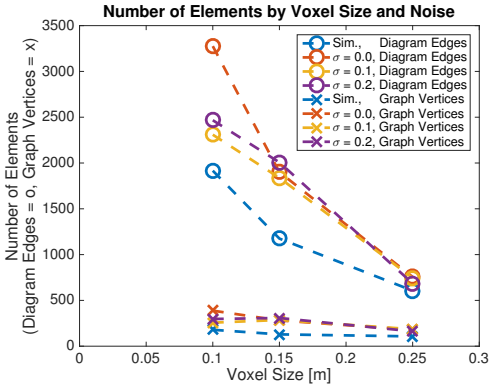



**Figure 8.8:** A comparison of the number of elements in the skeleton diagram (circles) and sparse graph (crosses). As can be seen, the voxel size and, to a lesser extent, noise level have a strong impact on the number of elements in the diagram, but the graph size stays mostly the same. This implies that the graph size is a function of the topology of the space, not the resolution or quality of the map.

## 6.2 Planning

To evaluate the ability of our method to very quickly generate initial feasible paths through complex environments, we set up a maze environment in simulation, and allowed a simulated MAV equipped with a forward-facing stereo camera to autonomously explore it using a next-best-view exploration algorithm [3]. The maze environment is 30 m by 30 m, represented by a map with 10 cm voxels. We then take the map created from this exploration and generate the ESDF, skeleton diagram, and sparse graph from it.

We use the methods described in Section 5 to plan between two locations in the maze, shown in Fig. 8.9. As can be seen, the sparse graph representation of the maze (red vertices and dark blue edges) appears to be a good descriptor of the underlying structure. While all the planning methods find largely similar paths, there are a few differences, since ESDF A* (yellow) and RRT* (green) are not bound to staying on the diagram (orange) or sparse graph (cyan).

However, the major difference is in the execution time, shown in Table 8.1. The RRT* takes approximately 2.5 seconds to find the initial solution, while an A* search through the complete ESDF takes over 7 minutes. In contrast, a search through the diagram takes only 50 ms, and the sparse graph search takes only 3 ms.

It is also interesting to look at the path lengths; clearly our proposed method does not produce the shortest paths. It does produce the maximum-clearance

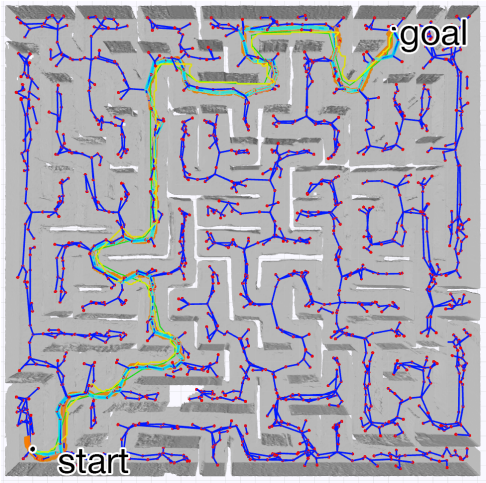



**Figure 8.9:** Sparse graph (dark blue edges and red vertices) and planning results through a 30 m by 30 m maze, autonomously explored by an MAV in simulation. Green shows the first RRT* path to the goal, yellow is A* through the ESDF, orange is A* through the skeleton diagram, and cyan is the search through the sparse graph.

| Planner | Time $[s]$ | Path Length $[m]$ | Solution Vertices |
|---|---|---|---|
| RRT* | 2.500 | **62.079** | **39** |
| RRT Connect | 0.1530 | 108.07 | 68 |
| A* ESDF | 452.3 | 72.26 | 627 |
| A* Skeleton Diagram | 0.0503 | 92.876 | 732 |
| Sparse Graph | **0.00328** | 86.178 | 110 |

**Table 8.1:** Quantitative results for planning through the maze, shown in Fig. 8.9. Our method (sparse graph) is 800 times faster than the initial solution of RRT*, and even 50 times faster than the RRT Connect. It produces slightly longer paths since it is bound to the maximum-clearance graph.

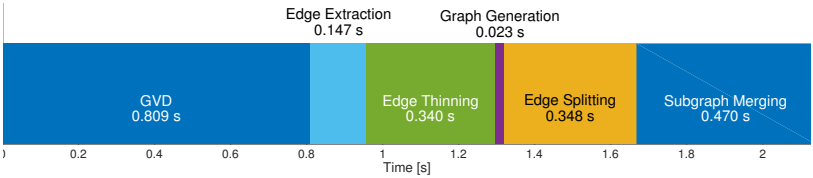



**Figure 8.10:** Timing information for the platform experiments, shown in Fig. 8.1. The map is 20 m by 22 m at 0.07 m voxel resolution. The total time to generate the sparse graph from the ESDF was 2.13 seconds on the MAV on-board computer.

path through this space, and can be further shortened by polynomial smoothing and optimization, as described in Section 5.5. The most important advantage of our method is that it takes almost 3 orders of magnitude less time to find this initial path using the sparse graph search than using RRT*.

## 6.3 Platform Experiments

To demonstrate the feasibility of this method on real data, we performed a flight through a machine hall with a custom-built drone, using ORB SLAM2 for state estimation and loop closure, and dense mapping of RGB-D data through the submapping approach described by Millane *et al.* [67]. This shows the intended use-case of this work: performing an initial exploratory flight with an MAV, quickly constructing a skeleton of the optimized map as shown in Fig. 8.10, and then using the topological map to rapidly plan return paths.

The results are shown in Fig. 8.1, where the sparse graph edges are shown in black, an initial path through the graph in yellow, and we use the techniques described in our previous work to get an initial dynamically-feasible polynomial path through a subset of the vertices (orange) and optimize it to be collision-free (red) [81, 85]. A video showing the experiments and results is available at `youtu.be/U_6rk-SFONw`.

This shows that we are able to quickly produce useful topological graphs to aid in global path planning from real sensor data from an MAV.

# 7 Conclusions

In this paper, we proposed a method to build 3D skeleton diagrams and sparse graphs that maintain the topology and connectivity of the original space, by using a combination of techniques from 2D Voronoi Diagram-based planning and 3D graphics skeletonization literature. Our method is robust to noise and resolution changes, and is shown to speed up global planning search queries by up to 800 times over initial solutions to RRT*. We also show its applicability to real maps built in-flight on an MAV, and demonstrate how our graph can be used to plan

optimized trajectories based on the fast initial solution. Future work would include adding clearance information to the sparse graph and making the method completely incremental.

Paper VI

# A Complete System for Vision-Based Micro-Aerial Vehicle Mapping, Planning, and Flight in Cluttered Environments

Helen Oleynikova, Zachary Taylor, Alexander Millane, Roland Siegwart, and Juan Nieto

**Abstract**

We present a complete system for micro-aerial vehicle autonomous navigation from vision-based sensing. We focus specifically on mapping using only on-board sensing and processing, and how this map information is best exploited for planning, especially when using narrow field of view sensors in very cluttered environments. In addition, details about other necessary parts of the system and special considerations are presented. We compare multiple global planning and path smoothing methods on real maps made in realistic search and rescue and industrial inspection scenarios.

Published in:
*arXiv:1812.03892*, 2018

## 1 Introduction

Autonomous navigation from on-board sensing is essential for Micro-Aerial Vehicles (MAVs) in many applications. Specifically, we want to create MAVs that can assist human operators in difficult inspection tasks in search and rescue (S&R) and industrial applications. To address both of these needs, we do not use GPS, and focus on online mapping and planning from only vision-based sensors.

This paper aims to present a complete system capable of performing repeated inspections of the same scene. Our previous work proposes mapping [83] and online re-planning [81, 85] methods for safe navigation of previously-unexplored spaces. We focus specifically on explicitly mapping free space in very cluttered environments, and exploring planning strategies that are inherently conservative: that is, they only allow traversal of space that is confirmed to be free. Here we aim to extend the local planning work to also cover global planning scenarios, where a map is already available (either on the return route of the current mission or from a previous mission).

We compare various global planning methods, including improving on our previously-proposed topological graphs built from Euclidean Signed Distance Fields (ESDFs) [86]. We also extend our local replanner to be used as a path-smoothing method for converting lists of waypoints from global planners to dynamically-feasible timed trajectories. All planning methods are evaluated on three realistic scenarios, two from a search and rescue training area, and one from an industrial environment with large machinery, recorded with the MAV system described in this paper.

The aim of this work is to serve as a reference for the *complete system* needed for these applications, requiring no off-board processing or external sensing. This set-up allows the robot to be robust to loss of communication (which is typical in real S&R scenarios), and behave intelligently and safely even when not under direct control of a human. We discuss the control, state estimation, sensor, and hardware concerns and requirements for such systems, and make the software for all parts available open-source.

The contributions of this work are as follows:

- We present a complete open-source system for autonomous GPS-denied navigation,
- A thorough treatment of considerations in 3D mapping for planning applications, expanding on our previous work [83] with regards to unknown space and generation methods,
- We extend our previous work on topological sparse graph generation [86] to create more useful graphs, faster,
- A discussion and comparison of various global planning methods,
- We extend our previous work in local planning [81, 85] to also be usable for path smoothing and compare to several competing methods.

**Figure 9.1:** The platform used for collecting the test datasets, a custom-built drone using the DJI FlameWheel F550 frame, an Intel NUC for on-board processing, Pixhawk for flight control, VI Sensor for monocular state estimation and stereo depth, and an Intel RealSense D415 for RGB-D input.

## 2 Related Work

We will give a very abbreviated overview of related work, as more thorough discussions of all parts are available in our previous work [81, 83, 85, 86].

We aim to show a complete system for mapping and planning on-board an autonomous UAV, using vision-based sensing. Lin *et al.* [57] presented a similar complete system, spanning visual-inertial state estimation, local re-planning, and control. However, there are a few key differences between the frameworks proposed: ours focuses strongly on the map representation we use and exploiting all the information within, while their uses a standard occupancy map. More importantly, our planning is *conservative*, meaning we will only traverse known free space, while theirs assumes unknown space is free. Therefore, we must make more considerations about the contents of our map with these restrictive assumptions. We also offer an evaluation of global and path smoothing planning methods.

Mohta *et al.* [69] also propose an autonomous system for fast UAV flight through cluttered environments. There are a few key differences with their work, especially on mapping and planning. They use a LIDAR as the main sensor, which gives 360° field of view for collision detection, removing many of the issues with narrow field of view sensors which we attempt to address in this work. They also only keep a small local 3D map, and use a global 2D map to escape local minima, whereas we use a full global 3D approach at comparable computation speeds. For how the mapping is used, they attempt to break the world into overlapping convex free-space regions, which grows in complexity and is increasingly more limited as the space gets more complex, while we always plan directly in the map space. They also make no considerations for how drift will affect the map other than to keep

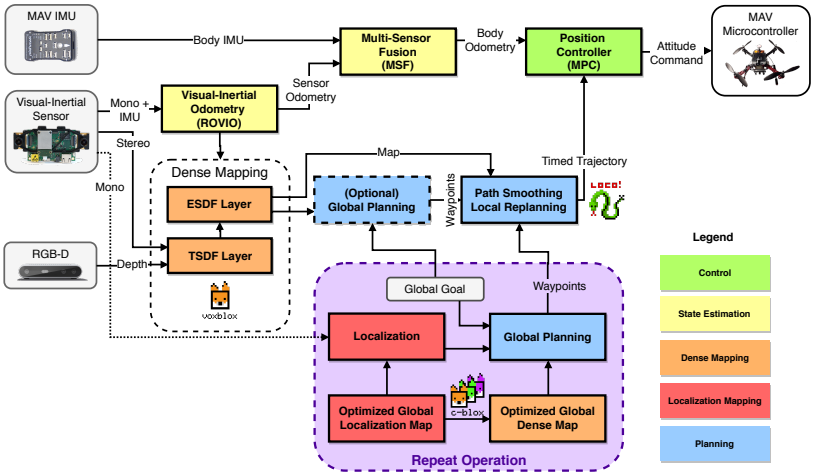



**Figure 9.2:** Overall system architecture, showing most key components of the system, and the data flow between mapping and planning. A second use-cases is shown in purple, where a previously-optimized global map is available for a mission.

only a local 3D map.

Finally, the system we propose is conceptually similar to the original system in our previous work [8]. The core differences are that we improved every individual component, designed and evaluated a custom mapping system, and proposed a way to do local re-planning as well (whereas the previous work was only global planning). This makes the system proposed in this work much more robust and able to deal with changes in the environment.

## 3 System Overview

We describe a complete MAV hardware and software system capable of supporting autonomous flight with only on-board vision-based sensing. All of the software described in the system has available open-source with provided links.

### 3.1 Overall Architecture

We show an overview of the complete system in Fig. 9.2, focusing on the data flow between mapping and planning processes. Stereo and depth images can be used interchangeably for the mapping, combined with a pose estimate from visual-inertial monocular odometry. The odometry is then fused with the body IMU of

the MAV to create a high-rate pose estimate used for control. Position control runs on the on-board computer, and gives roll, pitch, and yaw-rate commands to the attitude controller on the flight controller. The output of the planning stack are timed trajectories, sampled at the position controller rate (typically 100 Hz). Since we use a model-predictive controller (described in more detail below), this allows us to have a much higher trajectory tracking accuracy due to the long control horizon.

The diagram shows operation set up over two stages: the first is online flight through completely or partially unexplored space, in white, and in purple global planning in previously-built maps. This system allows us to do an initial flight, optimize a map using off-line tools, then perform repeat inspections in the same area. Using a local planner on the output of the global planners guarantees that even if the environment changes between missions, we are able to safely navigate through it by replanning locally. Likewise, we can use the same tools without stopping to optimize a map – for instance, using a global planner to return home at the end of a mission.

## 3.2 Hardware

Fig. 9.1 shows the MAV used to collect the evaluation datasets and test the overall system. It is built around a DJI FlameWheel F550, with 6 motors for actuation. The low-level control is performed with a pixhawk[1], using custom firmware that accepts attitude and yaw-rate commands [2]. This is necessary as we do not use GPS and fly indoors and in other environments where magnetometer readings are unreliable, therefore giving an absolute yaw reference is both undesirable and meaningless in our local coordinate frame.

On-board processing, which runs everything shown in Fig. 8.2, is performed on an Intel NUC. The main sensor is a custom-made visual-inertial sensor [78], with two monochromatic cameras in a stereo configuration, hardware-synced to an ADIS448 IMU. It is used for mono visual-inertial state estimation, and also for stereo depth for mapping. Optionally, we also use an Intel RealSense D415 for an additional source of RGB-D depth.

Though a dedicated visual-inertial sensor is a nice-to-have for such platforms, there is currently not one available off the shelf that is suitable for MAV flight. Instead, we recommend using a USB machine vision camera, and hardware time-synchronizing it to a flight controller. We make a sample driver implementing this for the FLIR Blackfly or Chameleon 3[3] and the pixhawk available[4]. This set-up is also extendable to multi-camera systems, as multiple cameras can be triggered from the same pulse.

[1] pixhawk.org
[2] github.com/ethz-asl/ethzasl_mav_px4
[3] ptgrey.com/blackfly-usb3-vision-cameras
[4] github.com/ethz-asl/flir_camera_driver

## 3.3 Control

We use a cascaded control architecture, with an inner loop that controls attitude and runs at a minimum of 100 Hz on the MAV autopilot, and an outer position control loop that runs on the on-board computer at 100 Hz. For the outer loop, we use a non-linear Model Predictive Control (MPC), proposed by Kamel *et al.* [42] and available open-source[5].

The MPC takes in the body odometry estimates from MSF (described in Section 3.4) and a timed full state trajectory to track. We exploit the properties of the flat state for MAVs to only need to specify position, yaw, and their derivatives [66].

One of the advantages of using an MPC over a PID loop for trajectory tracking is that the MPC is able to look ahead at future trajectory points, and minimize tracking error over the complete horizon. This means that overall trajectory tracking performance is improved significantly, and there are advantages to planning high-fidelity, dynamically-feasible trajectories, as they will be executed almost perfectly.

The non-linear MPC also has a *very* long horizon of 3 seconds, or 300 timesteps. While this is very convenient for executing long complex global trajectories, special care must be taken when using it for online replanning. Namely, we need to timestamp our entire trajectory to be monotonically increasing, and trajectory updates must be inserted into the correct place in the MPC queue. The queue is cleared if a trajectory with a time before the current execution time is sent, and the new trajectory replaces the complete queue. Fig. 9.3 shows an example of a replan cycle, happening for the purposes of the illustration at 50 Hz. The MPC queue is initialized with a starting trajectory. The local replanning "locks" the beginning of the initial trajectory, including the first 20 ms which is when the controller will receive the updated trajectory, and also another 30 ms look-ahead so that the reference does not change too quickly, then replans starting at 50 ms. A 3 second chunk, starting at the 50 ms, is then sent to the controller queue, which inserts the updated trajectory at the correct time, even though it has only executed up to 20 ms.

This queueing scheme allows us to replan at any given rate, while making sure that the controller always has a reasonable trajectory within its time horizon.

## 3.4 State Estimation

All state estimation is done on-board and not using external sensing (i.e., no vicon or GPS). This gives our system flexibility to be used in complex GPS-denied environments. Our main state estimator is Rovio[6], which is a robust visual-inertial odometry framework [6], [7]. Rovio is a filter-based estimator, which uses direct photometric error on a small number of patch features in the image. For our application, this design has a few distinct advantages over more traditional, keypoint-based methods like Okvis [56]: a filter with a small state-space (using only 25 features) is fast to compute, even on the on-board CPU, and using direct

[5] github.com/ethz-asl/mav_control_rw
[6] github.com/ethz-asl/rovio

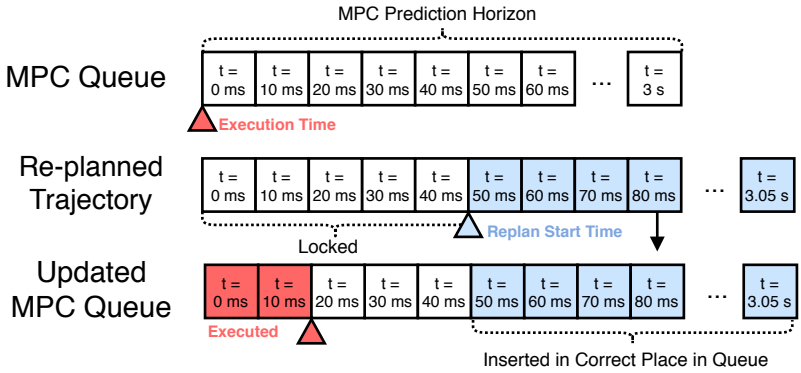



**Figure 9.3:** Diagram showing the MPC queue, and how it is updated when a section of the trajectory is replanned. Note that slightly more of the trajectory is locked down than is executed during the planning time. This allows the MPC to not have very abrupt changes in reference.

photometric error makes the method resistant to motion blur. In our experience, Rovio is comparably accurate to other methods such as Okvis and VINS-mono [92], but more robust under real conditions.

However, Rovio only gives us the odometry in the *sensor* frame, which for our system is usually the IMU that is part of the visual-inertial (VI) sensor. Rovio also only outputs updated poses at the camera frame rate. Depending on the hardware set-up, we have two solutions to receive body-IMU-frame odometry at 100 Hz.

If using a separate VI sensor, then the odometry estimate from Rovio must be transformed into the body frame and fused with the body IMU. This is done using Multi-Sensor Fusion (MSF) [62], which is a highly-configurable filter capable of taking multiple sensors and pose sources[7].

In another configuration, where the only IMU on the system is hardware time-synchronized to a camera, no transformation or fusion needs to be done. We only need to use the estimated Rovio biases to propagate the odometry estimate using incoming IMU measurements. This is done using a package called `odom_predictor`, which queues incoming IMU measurements and re-applies them when new (delayed) estimates are available from Rovio. It is also available open-source[8].

[7] github.com/ethz-asl/ethzasl_msf
[8] github.com/ethz-asl/odom_predictor

## 3.5 Localization

This paper aims to address issues with creating dynamically-feasible global plans. However, global planning requires *global localization*. Since all visual- and visual-inertial odometry frameworks drift, no matter how little, long-term operation or operation in previously-explored environments requires the ability to perform loop-closures.

To localize against a global map, we use maplab [98], an open-source[9] framework for creating, storing, optimizing, and localizing in visual-inertial maps.

To create global maps, we first generate a sparse pose-graph of landmarks in the observed scene. This is done by using Rovioli [98], a front-end for Rovio that also does feature tracking and extraction (independently of Rovio's 25 tracked patch features).

This sparse graph can then be loop-closed and optimized using bundle adjustment in an offline process, giving optimized, globally-consistent poses for all keyframes. To generate the global map, we can then replay all pointclouds from the initial flight and integrate them into a dense map using optimized poses. Localization in the matching sparse map will then line up correctly with the optimized dense map.

If using ORB-SLAM [73] as the SLAM system, a better approach is to build up a dense map using submaps, and only fuse them when the covariance between their relative poses is small enough, as presented in our previous work [68]. This allows us to get a globally-optimal dense map in a single step, without having to run a recorded dataset through an offline framework. However, since ORB-SLAM does not support localization against a previous map, this limits us to global localization within a single mission.

# 4 Dense Mapping

Dense mapping is key to planning performance, as a plan can only be as good as the map. We use a flexible mapping framework called *voxblox*[10], introduced in our previous work [83]. The framework is centered around using Signed Distance Fields (SDFs), or voxel grids of distance values to surfaces. We use two different types of SDFs: Truncated Signed Distance Fields (TSDFs), based on Curless and Levoy [16] and KinectFusion [74] for integrating point cloud data, in a method that gives a more accurate surface estimate than occupancy-based methods, but uses projective distances and truncates the value to a small band around surface crossings. The second type of field is a Euclidean Signed Distance Field (ESDF), which store Euclidean, rather than projective distances to each obstacle, and are not truncated to a specific range. These ESDFs are then the representation we use for planning, as they contain collision information for the entire map, and can also be used to quickly get obstacle gradients, which will be essential for some of the

[9] github.com/ethz-asl/maplab
[10] github.com/ethz-asl/voxblox

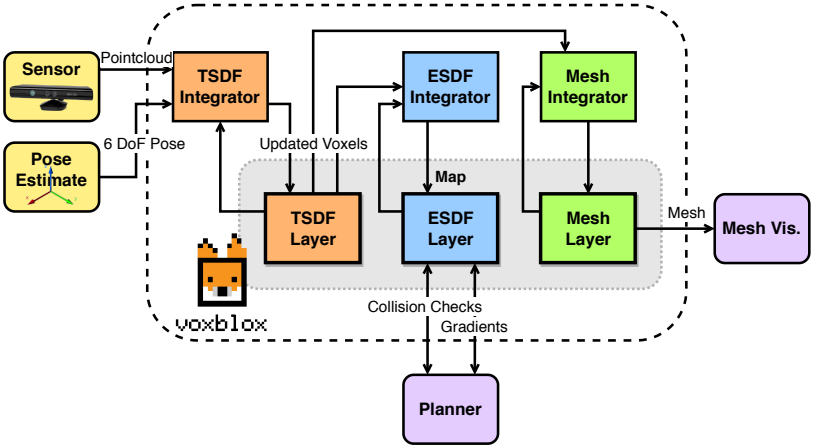



**Figure 9.4:** Voxblox system diagram, showing how the TSDF and ESDF layers are interconnected through integrators.

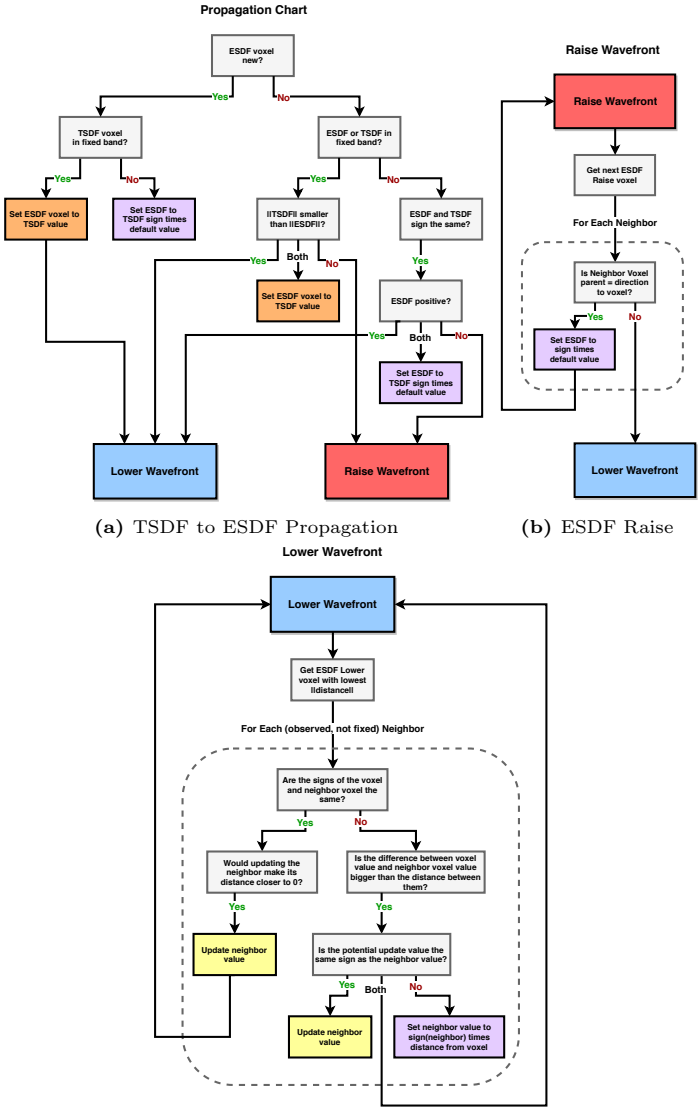
planning methods below. A system diagram showing inputs, outputs, and data flow is shown in Fig. 9.4.

## 4.1 Euclidean Signed Distance Fields

This section will discuss how an ESDF is computed from a TSDF. A more thorough analysis, including upper bounds on error introduced by various assumptions and a comparison with occupancy-based methods is offered in our previous work [83].

While it might seem unintuitive that there is a non-trivial process to convert from a TSDF to an ESDF, this is because the distances in both representations are computed differently. TSDFs use projective distance, or distance along the ray cast from the sensor to the surface. These distances are fairly accurate near surface crossings, but quickly accumulate large errors [82]. In contrast, an ESDF needs true Euclidean distances, which can only be calculated in a global fashion. Luckily, incremental algorithms exist for computing ESDFs from occupancy maps [51], and our work extends these methods to also work from TSDFs.

Generating the ESDF is done in three stages, detailed in Fig. 9.5: propagation (a), raise (b), and lower (c). The first, and most different from occupancy-based methods such as [51], is the TSDF to ESDF propagation. Due to the inaccuracy of TSDF distance estimates, we define a radius called a "fixed band" around the surface, which must be at least one voxel size and at most equal to the truncation distance. TSDF values that fall within this band are considered fixed, copied into the ESDF, and can not be altered in the ESDF update. Next, updated voxels may




**(a)** TSDF to ESDF Propagation **(b)** ESDF Raise

**(c)** ESDF Lower

**Figure 9.5:** Diagrams explaining how the ESDF is constructed in three stages.

simply retain their value, or be put into the "lower" wavefront (when their updated distance values are closer to the surface than before) or the "raise" wavefront (when their values become farther from the surface). If performing a batch update (i.e., the entire layer at once), all voxels will go to the lower wavefront.

After propagating all updated values from the TSDF into the ESDF, we then process the raise wavefront. This consists of simply invalidating all voxels in the wavefront and their children. Since each voxel stores its "parent" (if the voxel is in the fixed band from the TSDF, it is its own parent), this is then an incremental brushfire operation. All voxels cleared from the raise wavefront have their still-valid neighbors added to the lower wavefront, to guarantee that their values get updated.

The lower wavefront behaves similarly to the occupancy case: iterate over all voxels in the lower wavefront, if their neighbor's distance to the surface can be lowered through the current voxel, then update the neighbor and add it to the lower wavefront. Special distinctions must be made when implicit zero-crossings exist: i.e., two voxels are neighbors with opposite signs, and neither is fixed. This case is further explained in Fig. 9.5c.

## 4.2 Unknown Space

A key problem with mapping for collision avoidance is deciding how unknown space is handled. There are two options: treating all unknown space as free (optimistic), and treating all unknown space as occupied (pessimistic or conservative).

While many local collision avoidance works treat unknown space as free and have a high replan rate to avoid collisions [14], this is inherently unsafe. While it works well in uncluttered environments, where most unknown space *is* free, this assumption gets progressively worse as obstacle density in the environment increases and sensor field of view decreases.

Since our work aims to deal with the worst possible case, which is very obstacle-dense environments and a narrow field-of-view sensor, we cannot adapt the optimistic strategy. In fact, we always plan to stop in known free space, to guarantee safety in partially-unexplored environments.

However, it is a challenge to encode unknown space information in the ESDF, as the ESDF integration requires TSDF distances to build on, and those are either positive or negative. There is the additional problem that the MAV has no knowledge of the state of its current position at start-up or take-off, as it has never observed the space it occupies at the start. Furthermore, if the sensor has a very narrow field of view, the space it perceives in front of the sensor may not be wide enough to fit the entire robot body, essentially paralyzing the robot to never move. Finally, it is not clear how to correctly treat voxels bordering unknown space in ESDF computation.

We propose a simple strategy to resolve these issues, originally proposed in our previous work [85]. The idea is to have two overlapping "spheres" centered around the current robot pose that are applied in the ESDF, shown in Fig. 9.6. The inner sphere is small and only slightly larger than the robot radius and is called the

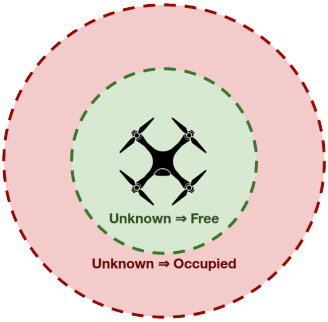



**Figure 9.6:** Diagram showing the two radiuses around the current robot pose: a small clear sphere, which should be only slightly larger than the robot, and an occupied sphere, which should be roughly the size of the planning radius.

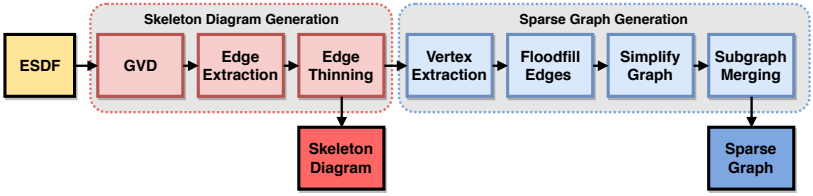



**Figure 9.7:** Steps in generating a sparse topological graph, also referred to as a skeleton graph.

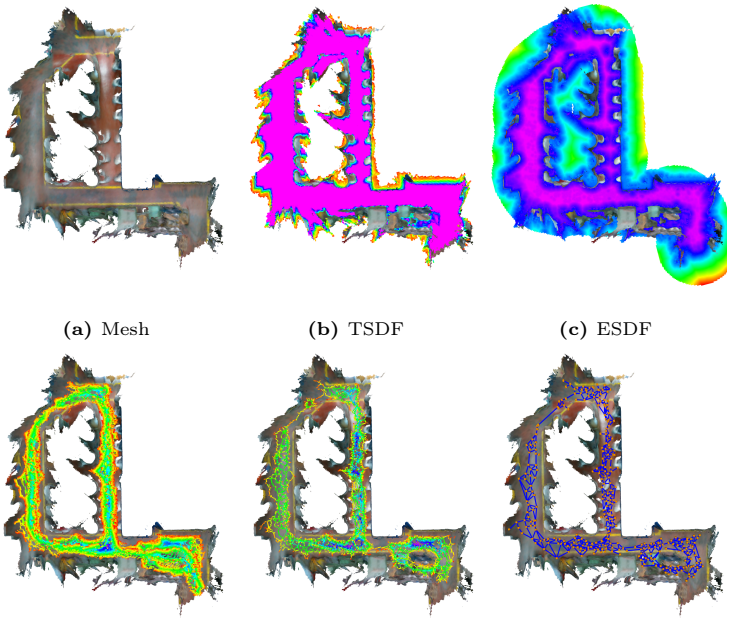
"clear sphere", which sets unknown space within it to free in the ESDF. The outer sphere affects all points not within the clear sphere and sets all unknown voxels to occupied. Any voxels that receive distances from this operation are marked as hallucinated in the map, so that as soon as real distance measurements are available from the TSDF, their values are overwritten.

## 5 Sparse Topology

In this section, we extend our work on generating sparse topological skeletons from [86]. We describe a complete method to extract a sparse graph of the traversable free space in an ESDF from only ESDF map data. This sparse graph is then used for very fast global planning in later sections of this work.

Our main contributions over our previous work in this section include:

- Switching to a simpler definition of the Generalized Voronoi Diagram (GVD),



**(a)** Mesh **(b)** TSDF **(c)** ESDF

**(d)** GVD **(e)** Thin GVD **(f)** Sparse Graph

**Figure 9.8:** Different stages of sparse topology skeleton computation, shown on the machine hall RealSense dataset. Colors represent distances from obstacles.

using 26-connectivity.

- Extending the method to work for both full and quasi-Euclidean distance.
- Speeding up sparse graph construction through use of flood-fill operations on the edges.
- Proposing a new sparse graph simplification and sub-graph re-connection methods, which produce more usable graphs significantly faster.

Fig. 9.7 shows the stages in the process, illustrated at key points with Fig. 9.8.

First, the generalized voronoi diagram (GVD) is constructed. This is done by iterating all voxels in the ESDF to find "ridges" or "basis points". A point is considered a basis point if its neighbors have parent vectors that are at least some angle apart. The angle differs depending on whether quasi-Euclidean or full Euclidean distance is employed, as quasi-Euclidean has a lower resolution in parent directions. For full Euclidean distance, the separation angle is 45°, and for quasi-Euclidean it is 90°.

We use a different definition of what belongs on the GVD from our previous work, making the definition simpler and more physically meaningful. Before we used 6-connectivity when evaluating belonging on the GVD, but now we use full 26-connectivity for all parts of the process. A point is considered a GVD face if it has 9 or more neighbors that are basis points, an edge if it has 12 or more, and a vertex at 16 or more. For the purposes of the remaining method, we do not consider faces, as the goal is to build as sparse of a diagram as possible.

Finally, to create a sparse diagram (which is simply a voxel layer containing number of basis point neighbors and whether the point is an edge or vertex or not), we apply a series of thinning operations described in more detail in [86]. Fig. 9.8d and e show the difference before and after thinning: after the operation is applied, all that is left is a one voxel-thick skeleton diagram.

To be more useful for planning, we want to further sparsify this diagram into a graph, removing the notion of discrete voxel sizes. We propose a different method of generating the sparse graph from the skeleton diagram here, which does not follow the underlying diagram exactly (as our previous work did) but greatly simplifies it. The downside to this is that edges no longer follow the maximum-clearance edges, but may now pass through intraversable space. However, they will always be *very near* traversable space, therefore as long as a flexible smoothing method is employed afterwards, a feasible path will be found.

We take all vertices in the skeleton diagram, assign them unique vertex IDs, and perform a flood-fill in all directions that contain edges, labelling the edges with the two nearest vertex IDs. This is a speed improvement and simplification over our previous "edge-following" algorithm, and does not suffer from cases where edge connections are missed.

To simplify this graph, we attempt to remove all vertices that are not adding information to the graph – essentially, vertices that are on straight lines or nearly-straight lines between other vertices. The filter consists of removing vertices that have exactly two edges, and whose removal will not displace the edges more than

2 voxels.

As a final step, we attempt to find any way to reconnect disconnected subgraphs. We label each self-contained subgraph in the sparse graph with another flood-fill operation, and assign subgraph IDs. We then iteratively search for connections from all subgraphs to all other subgraphs: if a connection is found, one of the subgraphs is relabelled. To see if two sub-graphs can be connected, we first search along the skeleton diagram using A*: if a connection exists, we insert a new edge between the two closest vertices. If no connection exists in the diagram (which very occasionally happens due to discretization error), we search in the traversable space of the ESDF, again using A*. If a path exists there, then we attempt to verify that ESDF path is valid and close to straight-line.

Both of these methods are much faster and more reliable at producing usable sparse planning graphs than our previous approach.

# 6 Global Planning

In this section we will discuss different global planning strategies and their advantages and disadvantages. It is assumed that a complete global map is available for these methods. We plan only in position space, assuming the MAV is a sphere for collision-checking purposes (as this makes it rotationally-invariant), and the output of all the global planners is a list of position waypoints from start to goal. All methods described here are available open-source[11].

## 6.1 Sampling-based Methods

The first class of methods we will consider are sampling-based methods. They are very well suited for large 3D problems, as they do not suffer from the same scaling problem as search-based methods. One of the most commonly-used classes of methods is Rapidly-exploring Random Trees (RRT) [54], where random points are sampled in the planning space (in our case, just 3D position space) and iteratively connected to a tree. Once the goal point is sampled, there exists a path from the start point (the root of the tree) to the goal.

One key advantage of this class of methods is that all that is required is a way to determine if a randomly-sampled state is valid or not, and a way to determine whether connections between states are valid. We propose two different methods to do this: one in the ESDF, which is pessimistic (assuming unknown space is occupied) and on the TSDF, which is optimisic (assuming unknown space is free). As long as they are coupled with a conservative/pessimistic local planner, either method can be used.

The look-up in the ESDF requires only a single point per pose (as the ESDF already stores the distance to the nearest obstacle, and therefore as long as that distance is larger than the robot radius, the point is feasible), while the TSDF look-up requires looking up an entire sphere and every voxel within. Additionally,

[11] github.com/ethz-asl/mav_voxblox_planning

for determining if motions between two states are valid, we use a ray-cast operation to not miss any potentially occupied voxels.

#### RRT-Connect

RRT-Connect is a very fast variant of the original RRT algorithm, which does a bi-directional search: growing a tree from both the start and the goal points [49]. One the downside, it does not optimize the paths (the algorithm terminates once a path is found), so they are often much longer than necessary in complex environments. For the purposes of our benchmarks, we treat this as the "first solution" time and solution length for RRT-based algorithms.

#### RRT*

RRT* combines the best of both A* (which is an optional planning algorithm) and random sampling to create a probabilistically-optimal planning solution [44] . This method samples new points and rewires the graph for shorter solutions, up until a time-limit is reached. There are other variants, such as Informed RRT* [30], which iteratively shrink the sampling space after an initial solution is found, similar to how the admissible heuristic is used in A*.

In general, this is the class of methods we prefer to use, as they give short, nearly-optimal paths, and it is possible to decide how to trade-off computation time versus optimality depending on application.

#### PRM and PRM*

For planning in a changing map, RRT-based methods are a very powerful tool, as they do not store any information from iteration to iteration. However, for global planning in a static map, this discards and replicates a large amount of sampling effort. Therefore, Probabilistic Roadmaps (PRMs) are suitable for many applications where the map remains fixed.

These approaches are similar to the topological graphs described below, in that they usually consist of two stages: building the roadmap, and then searching the roadmap for a solution. However, they suffer the same drawbacks as all probabilistically-optimal methods: it is not clear how much sampling is "enough" sampling, so it can only be decided by heuristics, and small openings or narrow corridors pose huge problems for PRM-based methods (as the chance of samples landing within them are small).

### 6.2 Topological Graphs

Our proposed method is a search through the sparse topological graph generated in Section 5. Unlike PRM-based methods, ours is deterministic and has a fixed computation time. Additionally, since the graph is based on the structure of the ESDF, which already encodes the geometry of the scene, it does not suffer from narrow corridor openings. The downside to this fixed execution and search time is

that while the graph will contain all topologically-distinct homotopies of the space, it is not guaranteed that the path length in the graph is the same as the shortest path length through the space. We perform graph shortening (described below) to attempt to overcome this issue, but it is possible that an incorrect homotope will be selected (though this can also be a problem in PRMs, depending on how the point distributions are sampled).

The method works as a two-stage search, starting from the sparse graph from Section 5. We first find the nearest sparse graph vertex to the start and goal by using a pre-computed k-D tree of vertices. Then we find a path through the graph using A*. Due to the small size of the graph, this is a very fast operation, so it is possible to solve the problem for multiple start and goal vertices from the k-D tree to attempt to find a better solution. Finally, we plan from the start pose to the start vertex using A* in the ESDF, and likewise for the goal. Since these distances are always short, this is also not an expensive process.

#### Graph Shortening

While planning through the sparse topological graph is extremely fast, the waypoints it produces aim to maximize clearance, not necessarily minimize absolute path length (path length on the *graph* is minimized) through all traversable space. For instance, even a straight-line path from A to B would zig-zag along the maximum clearance lines in the graph.

To overcome this, we use iterative path shortening in the ESDF. We attempt to short-cut between pairs of waypoints on the initial graph path in a binary search manner, checking for traversability in the ESDF map, shown in Fig. 9.9.

We first try to shortcut directly from start to goal; if the straight-line path is not traversable, we then split the waypoint list into two halves: front to middle and middle to back. Each half is then iteratively checked, whether the intermediate vertices can simply be removed; if not, it is further split into two halves.

We perform this full splitting procedure multiple times to ensure that no further shortening is possible. This is similar to what the OMPL library does with the RRT-planned paths, with the important distinction that our method is deterministic, while theirs randomly tries to connect pairs of waypoints. This means that ours does not need heuristics to know when it is terminated: once no more changes are made, the waypoint list is as shortened as possible. The randomized approach requires options such as maximum steps and maximum empty steps (steps that do not shorten) before terminating, which means that it may still be possible to shorten the graph at termination.

## 7 Path Smoothing

Path smoothing deals with taking a set of waypoints and converting them to a smooth, dynamically-feasible path. We present three methods we compare for these purposes: velocity ramp, polynomial, and our approach named Loco. We

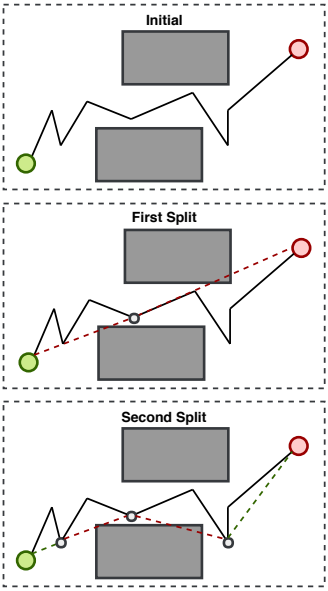



**Figure 9.9:** First three steps of binary-search graph shortening. The path is iteratively split in half, until sub-paths are able to be shortened or no more splits are possible. Green shows successful shortened paths, while red shows invalid shorten attempts.

enforce dynamic constraints in the form of maximum velocity and acceleration limits.

## 7.1 Velocity Ramp

The simplest method is velocity ramp. A straight-line path is drawn between consecutive pairs of waypoints, and maximum acceleration is applied until the velocity limit is reached, at which point the acceleration is zero. The same principle is applied on deccelerating toward the next point.

The total time between two waypoints is described as:

$$t = \frac{v_{max}}{a_{max}} + \frac{\left\| \mathbf{x}_{goal} - \mathbf{x}_{start} \right\|}{v_{max}} \tag{9.1}$$

where $t$ is the time in seconds, $v_{max}$ and $a_{max}$ are the velocity and acceleration constraints, respectively, and $\mathbf{x}_{start}$ and $\mathbf{x}_{goal}$ are the 3 DoF positions of the start

and goal.

## 7.2 Polynomial

High-degree polynomial splines are a common representation for MAV trajectories, as they are easy to compute, can be smooth and continuous up to high derivatives, and are shown to be dynamically feasible as long as velocity and acceleration constraints are met [66], [95].

We implement a path smoothing method from Richter *et al.* [95], where a polynomial spline is fit to the waypoints and then iteratively split at collisions.

First, we discuss the optimization problem. We formulate it to minimize a high derivative such as jerk or snap, as shown to be desirable by Mellinger *et al.* [66].

We will consider a polynomial spline in $K$ dimensions, with $S$ segments, and each segment of order $N$. Each segment has $K$ dimensions, each of which is described by an $N$th order polynomial:

$$f_k(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \dots a_N t^N \tag{9.2}$$

with the polynomial coefficients:

$$\mathbf{p}_k = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_N \end{bmatrix}^\top . \tag{9.3}$$

Rather than optimizing over the polynomial coefficients directly, which has numerical issues at high $N$s, we instead optimize over the end-derivatives of segments within the spline [95]. We distinguish between fixed derivatives $\mathbf{d}_F$ (such as end-constraints) and free derivatives $\mathbf{d}_P$ (such as intermediate spline connections):

$$\mathbf{p} = \mathbf{A}^{-1}\mathbf{M} \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix} . \tag{9.4}$$

Where $\mathbf{A}$ is a mapping matrix from polynomial coefficients to end-derivatives, and $\mathbf{M}$ is a reordering matrix to separate $\mathbf{d}_F$ and $\mathbf{d}_P$.

We aim to minimize the derivative cost, $J_d$, which represents a certain derivative (often jerk or snap) of the position [66], with $\mathbf{R}$ as the augmented cost matrix.

$$\begin{aligned} J_d \quad = \quad & \mathbf{d}_F^\top \mathbf{R}_{FF} \mathbf{d}_F + \mathbf{d}_F^\top \mathbf{R}_{FP} \mathbf{d}_P + \\ & \mathbf{d}_P^\top \mathbf{R}_{PF} \mathbf{d}_F + \mathbf{d}_P^\top \mathbf{R}_{PP} \mathbf{d}_P \end{aligned} \tag{9.5}$$

Finding the $\mathbf{d}_P^\star$ that minimized $J_d$ is possible to do in closed-form [95]:

$$\mathbf{d}_P^\star = -\mathbf{R}_{PP}^{-1} \mathbf{R}_{FP}^\top \mathbf{d}_F \tag{9.6}$$

This method allows us to fit a smooth polynomial spline to a series of waypoints, by using the positions of the waypoints as vertex constraints.

However, since waypoint trajectories are planned such that the *straight-line* path (visibility graph) between them is collision-free, the smoothed path often runs into

collision. To remedy this, any time a collision is detected, this method adds a new waypoint on the visibility graph closest to its projection onto the straight-line path, and the optimization is re-run.

While this is fast and easy to implement, this method suffers from occasionally not being able to escape collisions, and in difficult cases, creates many extra waypoints. The optimization problem does not scale well numerically when there are many waypoints, especially close together in time, as the segment times get very short (and must be raised to high powers). Furthermore, adding these additional waypoints often perturbs the trajectory in unexpected ways, causing the robot to take large detours.

## 7.3 Local Continuous Optimization (Loco)

To overcome these issues, we propose our own method, Local Continuous Optimization method, Loco~[81]. Rather than iteratively collision-checking and splitting the trajectories, we introduce the collisions as soft costs in the optimization, following the general structure that Ratliff *et al.* ~[93] proposed in their CHOMP method.

Introducing this soft cost leads to the following optimization problem, where $w$ terms are constant weights:

$$\mathbf{d}_P^* = \underset{\mathbf{d}_P}{\operatorname{argmin}} \; w_d J_d + w_c J_c \tag{9.7}$$

$J_d$ remains as in (9.5) above, and we introduce a new term, $J_c$ represents a soft collision cost:

$$J_c = \sum_{t=0}^{t_m} c(\mathbf{f}(t)) \, \|\mathbf{v}(t)\| \, \Delta t \tag{9.8}$$

which approximates the line integral of costs along the path, where $c(\mathbf{x})$ is the collision cost from the map, $\mathbf{f}(t)$ is the position along the trajectory at time $t$, and $\mathbf{v}(t)$ is the velocity at time $t$.

For the collision cost in the map, we use a smooth gradually decreasing function proposed in CHOMP~[93], where $\epsilon$ is a tuning value for how far outside the robot radius we care about collisions, $\mathbf{x}$ is a position in the map, and $d(\mathbf{x})$ is the ESDF distance at that point:

$$c(\mathbf{x}) = \begin{cases} -d(\mathbf{x}) + \frac{1}{2}\epsilon & \text{if } d(\mathbf{x}) < 0 \\ \frac{1}{2\epsilon}(d(\mathbf{x}) - \epsilon)^2 & \text{if } 0 \leq d(\mathbf{x}) \leq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{9.9}$$

In practice our robot radius is rarely 0, so we subtract the robot radius $r$ from $d(\mathbf{x})$.

Now that we have a method of locally optimizing trajectories to be collision-free, evaluated at length in our previous work~[81], the question is how to best make it fit through a series of waypoints.

We can use the starting method described above, where each waypoint simply becomes a control point/vertex in the spline. This means that for every waypoint, we have another segment in the spline. This has various downsides, most notably that for long, complex trajectories, the problem gets significantly slower computationally, and can have numerical scaling issues.

We found in practice that the optimization works best with a small (3-5) number of segments, therefore we explored methods to fit a visibility graph to these segments. The first solution was to generate an initial polynomial solution passing through all waypoints, and then re-sample it down to $S$ segments, by selecting $S-1$ evenly-spaced times to sample the trajectory at. These then become the new waypoints. We will refer to this strategy as "polynomial resampling" in the results in Section 8.3.

The second method is to instead sample directly on the visibility graph. Rather than sampling evenly-spaced $t$s on a polynomial trajectory, we instead fit a velocity ramp straight-line trajectory to a visibility graph, and sample the $t$s on this trajectory. This method is referred to as "visibility resampling" in the results.

As shown in Section 8.3, both of these methods create better, higher-quality paths faster than simple waypoint fitting.

# 8 Evaluations

We attempt to validate our complete system, especially focusing on global planning and path smoothing on real data.

The local replanning is separately validated on both synthetic test-cases and in the real world in our previous work [81], [85].

## 8.1 Evaluation Datasets

We focus our evaluations on real datasets, collected in typical scenarios for search and rescue and industrial inspection. We performed three inspection flights, two at a military search and rescue training ground at Wangen an der Aare, and one in the ETH Zürich Machine Hall. Photos of the three areas, named "shed", "rubble", and "Machine Hall" respectively, are shown in Fig. 9.10 and described in Table 9.1. All datasets were collected with the MAV and sensor set-up described in Section 3.2, using both stereo and RGB-D (Intel RealSense D415) sensors.

We make six global maps available: two per dataset, one using the stereo cameras and one using the RGB-D sensor (RS). All the maps are available for download[12].

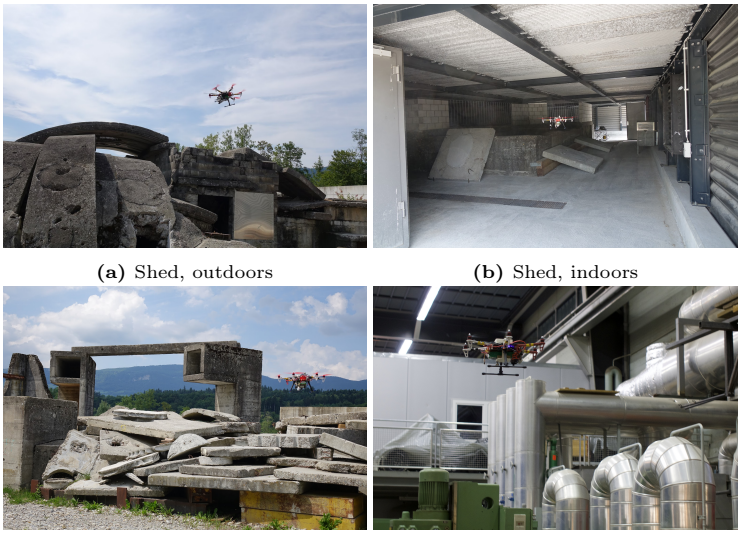
The provided maps were generated with 10 cm voxels, 1 meter clear and 4 meter occupied spheres (described in Section 4.2), 8 meter maximum ray distance for TSDF construction, and 4 meter maximum ESDF computation distance.

We provide both a stereo and an RS map of all environments due to the narrow field of view (FoV) of the RealSense camera, but superior depth measurements (and color information). Fig. 9.12 shows the difference in traversability between

[12] github.com/ethz-asl/mav_voxblox_planning

| Dataset Name | Location | Flight Length | Volume | Contents |
|---|---|---|---|---|
| Shed | Wangen a. A, BE, CH | 217 sec | 38 m x 35 m x 12 m | Mixed indoor and outdoor dataset with narrow openings |
| Rubble | Wangen a. A, BE, CH | 159 sec | 28 m x 27 m x 12 m | Outdoor dataset, over earth-quake damaged buildings |
| Machine Hall | ETH Zürich, ZH, CH | 251 sec | 24 m x 30 m x 8 m | Indoor area, with large industrial machinery |

**Table 9.1:** Dataset statistics and descriptions.



**(a)** Shed, outdoors **(b)** Shed, indoors

**(c)** Rubble **(d)** Machine Hall

**Figure 9.10:** Photos of the three scenarios from the benchmark dataset. (a) and (b) outdoor and indoor parts of the shed scene. (c): rubble scene. (d): machine hall scene.

the stereo and the RGB-D version of the shed dataset, assuming an 0.5 meter robot radius. For other datasets, such as machine hall, this is less important because the structure is much closer and therefore the narrow FoV makes little difference.

## 8.2 Sparse Topology Generation

To show that our sparse topology graph (or skeleton) is a feasible global planning strategy, we analyze the amount of time it takes to generate the sparse graph from an ESDF for each dataset. The results are shown in Fig. 9.13.

There is a large difference in the generation time between stereo and RealSense for the machine hall and rubble datasets, due to how much more space is traversable in the stereo datasets. However, most datasets are generated in 2 seconds or less, and the worst-case is 10 seconds. We consider this very feasible for a pre-processing step for global planning, as the actual planning times are orders of magnitude faster than other methods.

## 8.3 Loco

We analyze the effect of different waypoint fitting methods for Loco, as described in Section 7.3. We compare three methods: waypoint fitting (where each pair of waypoints has a segment between them), polynomial resampling (where an initial polynomial trajectory is fit with one segment through each waypoint, then resampled to a fixed number of waypoints), and visbility graph resampling (where intermediate waypoints are sampled directly off the visibility graph). The results are shown in Fig. 9.14, evaluated on the stereo shed dataset.

The visibility graph resampling has the highest success rate, and we use that variant as 'Loco' for the remaining evaluations.

## 8.4 Global Planning Benchmarks

To demonstrate the differences between different global planning methods, and how choice of path smoother affects the final result, we run 100 trials on each provided dataset. Each trial starts and ends at a random location, a minimum of 2 meters apart. The robot radius is 0.5 meters for all planners.

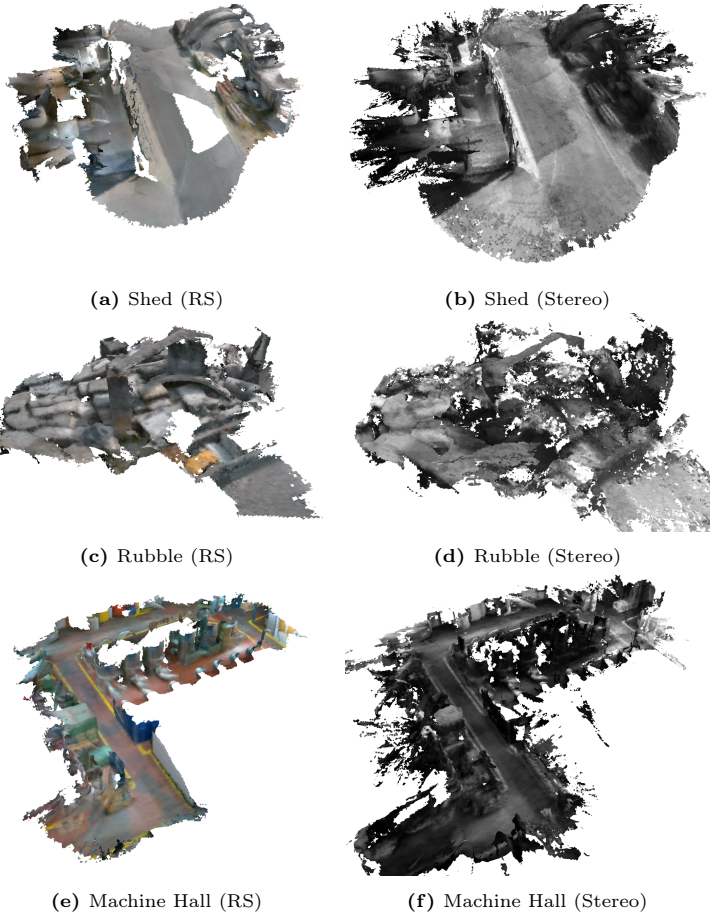
We use multiple global planning methods, summarized below:

**None** Straight-line path between start and goal, no planning, meant to give an estimate of how many of the test cases have trivial solutions.

**RRT Conn.** RRT-Connect, which grows a bi-directional tree from and toward the goal. Very fast, run with an upper time bound of 1.0 sec (though terminates when first solution is found).

**RRT*** Probabilistically-optimal random planner, run for 2.0 seconds.

**Skeleton** Our sparse topological planner, using path shortening on the output path.



**(a)** Shed (RS)

**(b)** Shed (Stereo)

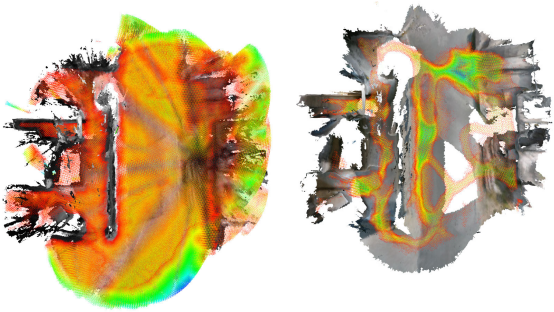
**(c)** Rubble (RS)

**(d)** Rubble (Stereo)

**(e)** Machine Hall (RS)

**(f)** Machine Hall (Stereo)

**Figure 9.11:** Colored meshes of the RealSense (RS) and stereo versions of the three maps used for benchmarking. Our stereo system is based on grayscale cameras, so no color data is available.



**(a)** Shed (Stereo) **(b)** Shed (RS)

**Figure 9.12:** Traversability differences between stereo datasets and RealSense datasets. Traversable space (given an 0.5 meter robot radius) is shown as colored points. As can be seen, much more space is considered traversable with stereo data.

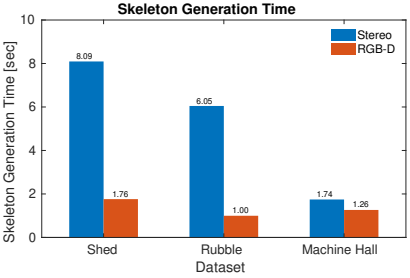



**Figure 9.13:** Sparse topological (skeleton) graph generation timings for the test datasets. The stereo datasets generally take longer because there is more traversable space.

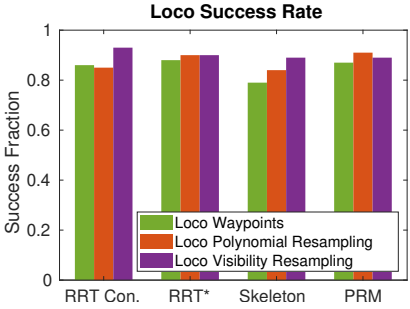



**Figure 9.14:** A comparison of different waypoint fitting methods for Loco, and their success rate on the shed stereo dataset given different global planning methods. In general, visibility resampling has the highest success rate.

**PRM** Probabilistic roadmap, aimed to compare versus our skeleton-based method. The pre-planning stage is run for 2.0 seconds (to mirror the average dataset processing time of the sparse topology), and each planning query is given an additional 0.1 seconds.

Likewise, we test a variety of path smoothing methods:

**No Smoothing** Not an actual path smoothing method, just an indicator showing whether the global planner succeeded or not.

**Velocity Ramp** Velocity ramp method, always applying maximum or no acceleration. Follows straight-line paths between waypoints.

**Polynomial** The polynomial splitting approach of [95], described in sections above.

**Loco** Our local continuous trajectory optimizaton algorithm, run with visibility waypoint re-sampling, as determined from the previous sections to be the best.

Fig. 9.15 shows a comparison of all described methods on the Machine Hall RealSense dataset. There are a number of take-aways from these results. When not using a global planner (i.e., attempting to draw a straight-line path between start and goal), only 13% of the test cases have trivial solutions, but Loco is able to solve 56% of problems with no global plan. In general, the success rate of Loco is also slightly higher, as it is able to better utilize the information in the map.

All the global planners are able to solve all of the planning problems. One key point to note is that the velocity ramp method does not work very well with the topological skeleton planner: this is because our graph simplification method

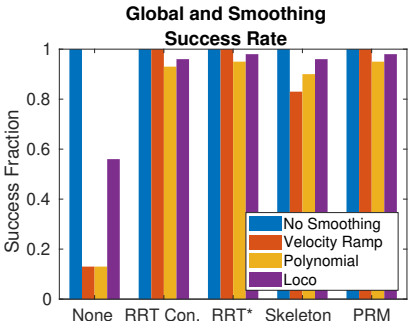



**Figure 9.15:** Success rate of various global planner and path smoothing methods on the Machine Hall RealSense dataset, showing our method (Loco) is able to give smooth dynamically-feasible paths for more tests cases than competing methods.

contains some edges that do not lie perfectly on the straight-line. However, the Loco planner has a comparable success rate with the skeleton planner as other planners, again because it can follow gradient information in the map and slightly perturb the waypoints to produce a collision-free path.

The timings for a single typical trajectory on the Shed stereo dataset are shown in Fig. 9.16 (note the log scale). The topological skeleton planning method is 10x faster than even RRT Connect (and produces much shorter path lengths). Both the polynomial and loco smoothing methods are acceptable for fast global planning, though the polynomial method is approximately 2x faster.

## 9 Conclusions

This paper presents a complete system for performing global planning, path smoothing, and local replanning. We extend on our previous work by improving a global planning sparse topology generation algorithm, suggest methods in which our local re-planning algorithm can also function for path smoothing, and benchmark a variety of global and path smoothing methods. Most importantly, we describe not only our mapping and planning approaches, but considerations that must be taken in other parts of the system for this approach to work, such as state estimation and controls, and make all of our code available online and open-source.

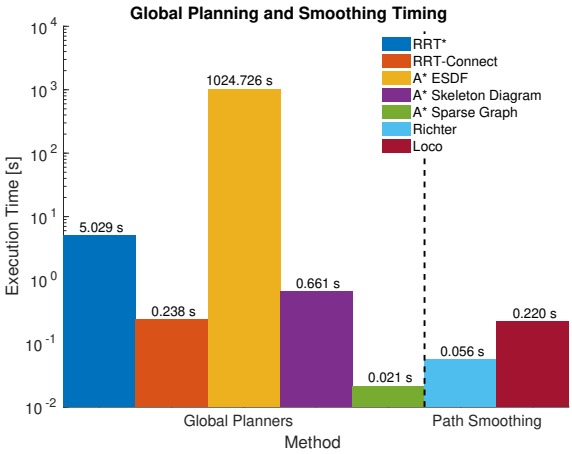



**Figure 9.16:** Timings for various global and local planning methods. Note the log scale. As can be seen, skeleton planning is at least one order of magnitude faster than other global planning methods, and while loco timing is slightly slower than polynomial, it is still within bounds for fast global planning applications.

# Bibliography


[1] A. J. Barry. *High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo*. PhD thesis, Massachusetts Institute of Technology, Feb 2016.

[2] A. J. Barry, A. Majumdar, and R. Tedrake. Safety verification of reactive controllers for uav flight in cluttered environments using barrier certificates. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 484–490. IEEE, 2012.

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

[4] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots*, pages 1–16, 2016.

[5] F. Blöchliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart. Topomap: Topological mapping and navigation based on visual slam maps. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

[6] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304. IEEE, 2015.

[7] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart. Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research (IJRR)*, 36(10): 1053–1072, 2017.

[8] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015.

[9] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research (IJRR)*, 2016.

[10] M. Burri, J. Nikolic, H. Oleynikova, M. W. Achtelik, and R. Siegwart. Maximum likelihood parameter identification for mavs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4297–4303. IEEE, 2016.

[11] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems (RSS)*, volume 9. Robotics: Science and Systems, 2013.

[12] T.-T. Cao, K. Tang, A. Mohamed, and T.-S. Tan. Parallel banding algorithm to compute exact distance transform with the gpu. In *ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 2010.

[13] B. Charrow, G. Kahn, S. Patil, S. Liu, K. Goldberg, P. Abbeel, N. Michael, and V. Kumar. Information-theoretic planning with trajectory optimization for dense 3d mapping. In *Robotics: Science and Systems (RSS)*, 2015.

[14] J. Chen, T. Liu, and S. Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2016.

[15] H. Cover, S. Choudhury, S. Scherer, and S. Singh. Sparse tangential network (spartan): Motion planning for micro aerial vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2820–2825. IEEE, 2013.

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

[17] B. Davis, I. Karamouzas, and S. J. Guy. C-opt: Coverage-aware trajectory optimization under uncertainty. *IEEE Robotics and Automation Letters*, 2016.

[18] R. Deits and R. Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI*, pages 109–124. Springer, 2015.

[19] J. Dong, M. Mukadam, F. Dellaert, and B. Boots. Motion planning as probabilistic inference using gaussian processes and factor graphs. In *Robotics: Science and Systems (RSS)*, June 2016.

[20] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[21] Z. Fang, C. Luan, and Z. Sun. A 2d voronoi-based random tree for path planning in complicated 3d environments. In *International Conference on Intelligent Autonomous Systems*, pages 433–445. Springer, 2016.

[22] P. Florence, J. Carter, and R. Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Workshop on the Algorithmic Foundations of Robotics*, 2016.

[23] P. R. Florence, J. Carter, J. Ware, and R. Tedrake. Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[24] M. Foskey, M. Garber, M. C. Lin, and D. Manocha. A voronoi-based hybrid motion planner. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 55–60. IEEE, 2001.

[25] M. Foskey, M. C. Lin, and D. Manocha. Efficient computation of a simplified medial axis. *Journal of Computing and Information Science in Engineering*, 3(4):274–284, 2003.

[26] D. Fox, W. Burgard, S. Thrun, and A. B. Cremers. A hybrid collision avoidance method for mobile robots. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1238–1243. IEEE, 1998.

[27] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.

[28] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. Rotors – a modular gazebo mav simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.

[29] F. Furrer, T. Novkovic, M. Fehr, A. Gawel, M. Grinvald, T. Sattler, R. Siegwart, and J. Nieto. Incremental object database: Building 3d models from multiple partial observations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

[30] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2014.

[31] S. F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *IEEE Symposium on Volume Visualization*, pages 23–30. IEEE, 1998.

[32] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[33] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2472–2477. IEEE, 2011.

[34] L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous visual mapping and exploration with a micro aerial vehicle. *Journal of Field Robotics*, 31(4):654–675, 2014.

[35] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research (IJRR)*, 31(5):647–663, 2012.

[36] C. Hernández, G. Vogiatzis, and R. Cipolla. Probabilistic visibility for multi-view stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007.

[37] K. Hoff, T. Culver, J. Keyser, M. C. Lin, and D. Manocha. Interactive motion planning using hardware-accelerated computation of generalized voronoi diagrams. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2931–2937. IEEE, 2000.

[38] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 2013.

[39] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr, and D. Murray. Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 2015.

[40] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011.

[41] N. Kalra, D. Ferguson, and A. Stentz. Incremental reconstruction of generalized voronoi diagrams on grids. *Robotics and Autonomous Systems*, 57(2): 123–128, 2009.

[42] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto. Nonlinear model predictive control for multi-micro aerial vehicle robust collision avoidance. *arXiv preprint arXiv:1703.01164*, 2017.

[43] M. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017.

[44] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research (IJRR)*, 30(7): 846–894, 2011.

[45] S. Karaman and E. Frazzoli. High-speed flight in an ergodic forest. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2899–2906. IEEE, 2012.

[46] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device. In *RSS: Robotics Science and Systems*, July 2015.

[47] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics (T-RO)*, 21(3):354–363, 2005.

[48] P. Krüsi, B. Bücheler, F. Pomerleau, U. Schwesinger, R. Siegwart, and P. Furgale. Lighting-invariant adaptive route following using iterative closest point matching. *Journal of Field Robotics*, 32(4):534–564, 2015.

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

[50] B. Landry, R. Deits, P. R. Florence, and R. Tedrake. Aggressive quadrotor flight through cluttered environments using mixed integer programming. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1469–1475. IEEE, 2016.

[51] B. Lau, C. Sprunk, and W. Burgard. Improved updating of euclidean distance maps and voronoi diagrams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010.

[52] B. Lau, C. Sprunk, and W. Burgard. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics and Autonomous Systems*, 61(10):1116–1130, 2013.

[53] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research (IJRR)*, 20(5):378–400, 2001.

[54] S. M. Lavalle, J. J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[55] T.-C. Lee, R. L. Kashyap, and C.-N. Chu. Building skeleton models via 3-d medial surface axis thinning algorithms. *CVGIP: Graphical Models and Image Processing*, 56(6):462–478, 1994.

[56] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research (IJRR)*, 2015.

[57] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen. Autonomous aerial navigation using monocular visual-inertial fusion. *Journal of Field Robotics (JFR)*, 2017.

[58] M. Liu, F. Colas, L. Oth, and R. Siegwart. Incremental topological segmentation for semi-structured environments using discretized gvg. *Autonomous Robots*, 38(2):143–160, 2015.

[59] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2017.

[60] B. T. Lopez and J. P. How. Aggressive 3-d collision avoidance for high-speed navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[61] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, volume 21, pages 163–169. ACM, 1987.

[62] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013.

[63] C. M. Ma and M. Sonka. A fully parallel 3d thinning algorithm and its applications. *Computer vision and image understanding*, 64(3):420–433, 1996.

[64] B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev. Path planning for non-circular micro aerial vehicles in constrained environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3933–3940. IEEE, 2013.

[65] J. Machado Santos, D. Portugal, and R. P. Rocha. An evaluation of 2d slam techniques available in robot operating system. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pages 1–6. IEEE, 2013.

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

[67] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena. Tsdf manifolds: A scalable and consistent dense mapping approach. *arXiv preprint arXiv:1710.07242*, 2017.

[68] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena. C-blox: A scalable and consistent tsdf-based dense mapping approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

[69] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, et al. Fast, autonomous flight in gps-denied and cluttered environments. *Journal of Field Robotics (JFR)*, 35 (1):101–120, 2018.

[70] U. Montanari. A method for obtaining skeletons using a quasi-euclidean distance. *Journal of the ACM (JACM)*, 1968.

[71] B. Morrell, R. Thakker, G. Merewether, R. Reid, M. Rigter, T. Tzanetos, and G. Chamitoff. Comparison of trajectory optimization algorithms for high-speed quadrotor flight near obstacles. *IEEE Robotics and Automation Letters*, 3(4):4399–4406, 2018.

[72] M. W. Mueller, M. Hehn, and R. D'Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics (T-RO)*, 2015.

[73] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31 (5):1147–1163, 2015.

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

[75] C. V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*. IEEE, 2012.

[76] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 32(6):169, 2013.

[77] M. Nieuwenhuisen and S. Behnke. Layered mission and path planning for mav navigation with partial environment knowledge. In *Intelligent Autonomous Systems*, pages 307–319. Springer, 2016.

[78] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart. A synchronized visual-inertial sensor system with fpga preprocessing for accurate real-time slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 431–437. IEEE, 2014.

[79] H. Oleynikova, M. Burri, S. Lynen, and R. Siegwart. Real-time visual-inertial localization for aerial and ground robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept 2015.

[80] H. Oleynikova, D. Honegger, and M. Pollefeys. Reactive avoidance using embedded stereo vision for mav flight. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[81] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran. Continuous-time trajectory optimization for online uav replanning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016.

[82] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart. Signed distance fields: A natural representation for both mapping and planning. In *RSS Workshop on Geometry and Beyond*, 2016.

[83] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

[84] H. Oleynikova, Z. Taylor, A. Millane, R. Siegwart, and J. Nieto. A complete system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments. *arXiv preprint arXiv:1812.03892*, 2018.

[85] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto. Safe local exploration for replanning in cluttered unknown environments for micro-aerial vehicles. *IEEE Robotics and Automation Letters*, 2018.

[86] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto. Sparse 3d topological graphs for micro-aerial vehicle planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

[87] J. Pan, S. Chitta, and D. Manocha. Fcl: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3859–3866. IEEE, 2012.

[88] C. Papachristos, S. Khattak, and K. Alexis. Uncertainty-aware receding horizon exploration and mapping using aerial robots. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[89] C. Papachristos, M. Kamel, M. Popović, S. Khattak, A. Bircher, H. Oleynikova, T. Dang, F. Mascarich, K. Alexis, and R. Siegwart. Autonomous exploration and inspection path planning for aerial robots using the robot operating system. In *Robot Operating System (ROS)*, pages 67–111. Springer, 2019.

[90] A. A. Paranjape, K. C. Meier, X. Shi, S.-J. Chung, and S. Hutchinson. Motion primitives and 3d path planning for fast flight through a forest. *The International Journal of Robotics Research (IJRR)*, 2015.

[91] M. Pivtoraiko, D. Mellinger, and V. Kumar. Incremental micro-uav motion replanning for exploring unknown environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2452–2458. IEEE, 2013.

[92] T. Qin, P. Li, and S. Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[93] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009.

[94] A. Richards and J. P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the American Control Conference (ACC)*, volume 3, pages 1936–1941. IEEE, 2002.

[95] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2013.

[96] C. Richter, J. Ware, and N. Roy. High-speed autonomous navigation of unknown environments using learned probabilities of collision. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6114–6121. IEEE, 2014.

[97] F. Ruetz, E. Hernández, M. Pfeiffer, H. Oleynikova, M. Cox, T. Lowe, and P. Borges. Ovpc mesh: 3d free-space representation for local ground vehicle navigation. *arXiv preprint arXiv:1811.10266*, 2018.

[98] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart. maplab: An open framework for research in visual-inertial mapping and localization. *IEEE Robotics and Automation Letters*, 3(3): 1418–1425, 2018.

[99] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research (IJRR)*, 2014.

[100] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart. A sampling-based partial motion planning framework for system-compliant navigation along a reference path. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 391–396. IEEE, 2013.

[101] D. F. Shanno. On broyden-fletcher-goldfarb-shanno method. *Journal of Optimization Theory and Applications*, 46(1):87–94, 1985.

[102] F. She, R. Chen, W. Gao, P. Hodgson, L. Kong, and H. Hong. Improved 3d thinning algorithms for skeleton extraction. In *Digital Image Computing: Techniques and Applications (DICTA)*, pages 14–18. IEEE, 2009.

[103] S. Shen, N. Michael, and V. Kumar. Autonomous indoor 3d exploration with a micro-aerial vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–15. IEEE, 2012.

[104] F. Steinbrucker, J. Sturm, and D. Cremers. Volumetric 3d mapping in real-time on a cpu. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2021–2028. IEEE, 2014.

[105] A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, and A. Telea. 3d skeletons: A state-of-the-art report. In *Computer Graphics Forum*. Wiley Online Library, 2016.

[106] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research (IJRR)*, 2010.

[107] G. Teodoro, T. Pan, T. M. Kurc, J. Kong, L. A. Cooper, and J. H. Saltz. Efficient irregular wavefront propagation algorithms on hybrid cpu–gpu machines. *Parallel computing*, 2013.

[108] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[109] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers. Real-time trajectory replanning for mavs using uniform b-splines and 3d circular buffer. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

[110] R. Wagner, U. Frese, and B. Bauml. 3d modeling, distance and gradient computation for motion planning: A direct gpgpu approach. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013.

[111] R. Wagner, U. Frese, and B. Bäuml. Real-time dense multi-scale workspace modeling on a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013.

[112] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended kinectfusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[113] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. Elasticfusion: Dense slam without a pose graph. In *Robotics: Science and Systems (RSS)*, 2015.

[114] C. Witting, M. Fehr, R. Bähnemann, H. Oleynikova, and R. Siegwart. History-aware autonomous exploration in confined environments using mavs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

[115] Q.-Z. Ye. The signed euclidean distance transform and its applications. In *International Conference on Pattern Recognition (ICPR)*, pages 495–499. IEEE, 1988.

[116] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research (IJRR)*, 2013.

# Curriculum Vitae

**Helen Oleynikova**
born February 2nd, 1990
living in Zürich, Switzerland

| | |
|---|---|
| 2015–2018 | *ETH Zurich, Switzerland*<br>Doctoral studies at the Autonomous Systems Lab; Supervised by Prof. Roland Siegwart |
| 2013–2015 | *ETH Zurich, Switzerland*<br>Master of Science in Robotics, Systems, and Control |
| 2011–2013 | *Google, Mountain View, California, USA*<br>Software Engineer, Street View Team |
| 2007–2011 | *Franklin W. Olin College of Engineering, Needham, Massachusetts, USA*<br>Bachelor of Science in Engineering with a Concentration in Robotics |
| 2003–2007 | *Dublin High School, Dublin, California, USA*<br>High School Diploma |