Inaugural-Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Landwirtschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

Active 3D Reconstruction for Mobile Robots

von

Emanuele Palazzolo

aus
Turin, Italien

Bonn 2020

Referent:
Prof. Dr. Cyrill Stachniss, University of Bonn, Germany

Korreferent:
Prof. Dr. Philippe Giguère, Laval University, Québec, Canada

Tag der mündlichen Prüfung: 19 Dezember 2019

Erscheinungsjahr: 2020

Angefertigt mit Genehmigung der Landwirtschaftlichen Fakultät der Universität Bonn

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenenen Hilfsmittel angefertigt habe; die aus fremdem Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum

(Unterschrift)

Zusammenfassung

Die Abbildung der Umgebung durch die Rekonstruktion dreidimensionaler Modelle wird konventionell durch Fachpersonal mit speziellem Messequipment wie Kameras oder terrestrischen Laserscannern durchgeführt. Diese Verfahrensweisen sind allerdings mit hohen Kosten und einem erheblichen Zeitaufwand verbunden. Der Einsatz von Robotern ermöglicht hingegen automatisierte Verfahren sowie die Bestimmung dreidimensionaler Modelle innerhalb von für den Menschen unzugänglichen Umgebungen und deren Verwendung für Verbraucheranwendungen. Die vollständig autonome Rekonstruktion von 3D-Modellen ist jedoch eine nicht-triviale Aufgabenstellung und Schwerpunkt vieler Forschungsarbeiten. Die in dieser Arbeit vorgestellten Techniken befassen sich mit offenen Problemstellungen zur aktiven Rekonstruktion von 3D-Modellen. Für eine automatisierte Rekonstruktion muss der Roboter zum einen selbständig die optimalen Aufnahmepositionen bestimmen und zum anderen die Beobachtungen zeitgleich zur Erkundung der Umgebung in ein Modell integrieren. In dieser Arbeit adressieren wir zunächst die echtzeitfähige Integration von Sensorbeobachtungen in dichte, dreidimensionale Modelle. Im zweiten Schritt bestimmen wir die optimalen Aufnahmepositionen des Sensors, um die unbekannte Umgebung so effizient wie möglich zu rekonstruieren. Anschließend präsentieren wir eine Methodik zur Rekonstruktion von 3D-Modellen, die im Gegensatz zu klassischen Ansätzen keine vollständig statische Umgebung voraussetzt. Mit der entwickelten Methodik zielen wir insbesondere auf langfristige Veränderungen innerhalb der Szene ab und präsentieren einen Ansatz zur deren echtzeitfähigen Identifikation und Integration in ein bestehendes 3D-Modell. Abschließend adressieren wir die Identifikation von dynamischen Elementen innerhalb der Szene sowie deren Behandlung während der Messung.

Im ersten Teil dieser Arbeit präsentieren wir unter Annahme einer statischen Umgebung eine Methodik zur Lösung der beiden erstgenannten Aufgabenstellungen, die eine echtzeitfähige Rekonstruktion von 3D-Modellen basierend auf einem kommerziellen RGB-D Sensor ermöglicht. Der vorgestellte Ansatz zeichnet sich insbesondere durch eine hohe Effizienz hinsichtlich der Laufzeit und des Speicherbedarfs aus. Darüber hinaus ist unsere Methode insbesondere robust gegenüber Szenen in denen strukturelle Merkmale unzureichend vertreten sind. Zusätzlich stellen wir eine Methodik zur iterativen Berechnung der nächstbesten Aufnahmepositionen vor, welche die aus den Messungen erhaltene Information maximiert. Der vorgestellte Ansatz ist für Kleindrohnen, auch *micro aerial vehicles (MAV)* genannt, optimiert und berücksichtigt deren spezifischen Beschränkungen.

Im zweiten Teil dieser Arbeiten präsentieren wir eine Methodik für nicht-statische Umgebungen und adressieren die beiden letztgenannten Aufgabenstellungen. Basierend auf einer kurzen Bildsequenz identifiziert unser Ansatz Regionen des 3D-Modells, die von langfristigen Veränderungen innerhalb der Szene betroffen sind. Die von uns vorgestellte Methodik läuft echtzeitfähig auf einem Roboter, der basierend auf dieser Information explizit die von Veränderungen betroffenen Regionen berücksichtigen kann. Abschließend präsentieren wir eine echtzeitfähige Methodik zur Rekonstruktion dreidimensionaler Modelle innerhalb dynamischer Umgebungen, die nicht-statische Elemente identifiziert und aus den Messungen herausfiltert.

Insgesamt leistet diese Arbeit mehrere Beiträge im Kontext der roboterbasierten Rekonstruktion dreidimensionaler Modelle sowie dem Umgang mit Veränderungen innerhalb der Szene. Im Vergleich zum aktuellen Stand der Technik ermöglichen die in unserer Arbeit vorgestellten Ansätze ein robusteres, echtzeitfähiges Tracking von RGB-D Sensoren sowie die Behandlung von dynamischen Elementen. Darüber hinaus präsentieren wir eine effizientere Technik zur Auswahl der Aufnahmepositionen für die Exploration der Umgebung mittels MAV sowie eine effiziente, echtzeitfähige Identifikation von Veränderungen innerhalb der Szene basierend auf 3D-Modellen aus Bilddaten, die wesentlich schneller ist als vergleichbare bestehende Methoden. In Bezug auf Robustheit und Effizienz erweitert unsere Methodik damit den aktuellen Stand der Technik.

Abstract

MAPPING the environment with the purpose of building a 3D model that represents it, is traditionally achieved by trained personnel, using measuring equipment such as cameras or terrestrial laser scanners. This process is often expensive and time-consuming. The use of a robotic platform for such a purpose can simplify the process and enables the use of 3D models for consumer applications or in environments inaccessible to human operators. However, fully autonomous 3D reconstruction is a complex task and it is the focus of several open research topics. In this thesis, we try to address some of the open problems in active 3D environment reconstruction. For solving such a task, a robot should autonomously determine the best positions to record measurements and integrate these measurements in a model while exploring the environment. In this thesis, we first address the task of integrating the measurements from a sensor in real-time into a dense 3D model. Second, we focus on *where* the sensor should be placed to explore an unknown environment by recording the necessary measurements as efficiently as possible. Third, we relax the assumption of a static environment, which is typically made in active 3D reconstruction. Specifically, we target long-term changes in the environment and we address the issue of how to identify them online with an exploring robot, to integrate them in an existing 3D model. Finally, we address the problem of identifying and dealing with dynamic elements in the environment, while recording the measurements.

In the first part of this thesis, we assume the environment to be static and we solve the first two problems. We propose an approach to 3D reconstruction in real-time using a consumer RGB-D sensor. A particular focus of our approach is its efficiency in terms of both execution time and memory consumption. Moreover, our method is particularly robust to situations where the structural cues are insufficient. Additionally, we propose an approach to compute iteratively the next best viewpoint for the sensor to maximize the information obtained from the measurements. Our algorithm is taylored for micro aerial vehicles (MAV) and takes into account the specific limitations that this kind of robots have.

In the second part of this work, we focus on non-static environments and we address the last two problems. We deal with long-term changes by proposing an

approach that is able to identify the regions that changed on a 3D model, from a short sequence of images. Our method is fast enough to be suitable to run online on a mapping robot, which can direct its effort on the parts of the environment that have changed. Finally, we address the problem of mapping fully dynamic environments, by proposing an online 3D reconstruction approach that is able to identify and filter out dynamic elements in the measurements.

In sum, this thesis makes several contributions in the context of robotic map building and dealing with change. Compared to the current state of the art, the approaches presented in this thesis allow for a more robust real-time tracking of RGB-D sensors including the ability to deal with dynamic scenes. Moreover, this work provides a new, more efficient view point selection technique for MAV exploration, and an efficient online change detection approach operating on 3D models from images that is substantially faster than comparable existing methods. Thus, we advanced the state of the art in the field with respect to robustness as well as efficiency.

Acknowledgements

A wise man once told me that pursuing a Ph.D. is not about the title, but about the journey. He was right. During these years I have learned a lot and I have grown both professionally and personally, but most importantly I have met amazing people, without whom this thesis would probably not exist. First of all, I would like to thank Cyrill Stachniss for being the best supervisor I could have. He always supported me during my time in Bonn, he taught me a lot and without him I would have surely quit my Ph.D. halfway through, out of frustration. Thanks to him, I had the opportunity to travel around the world, attend amazing conferences, meet the best people in our field and have tons of fun. I will always be grateful for the huge contribution he made on my work and my life.

I would also like to thank Philippe Giguère for agreeing to review this thesis. During his time in our lab we engaged in several fruitful discussions. Our joint efforts led to a paper submission, which constitutes a significant part of this work. I hope he enjoyed our collaboration as much as I did, on both a professional and personal level.

I would like to extend my gratitude to all the people in the lab. They are not only colleagues, but also friends. I enjoyed the company of Jens Behley, Igor Bogoslavskyi, Xieyuanli Chen, Nived Chebrolu, Mathias Hans, Thomas Läbe, Philipp Lottes, Andres Milioto, Lorenzo Nardi, Johannes Schneider, Ignacio Vizzo, Olga Vysotska, and Jan Weyler inside and outside the lab. They were always helpful and ready to support me whenever I needed it. Also, a very big thanks to Birgit Klein, for helping me with any bureaucratic matter whenever I needed it, both related to work or personal. Without her, I would still be stuck figuring out German bureaucracy.

During my time in Bonn I met several interesting people. Without all of my friends I would not have been able to enjoy so much these years. They are too many to list them all, but I would like to particularly thank my best friend Elena Nardi. She is literally *always* ready to help me, support me and listen to my complaints in front of a beer (or many). We had a lot of fun together and I could not have survived the process of writing this thesis without her. Second, I would like to thank Gwydyon Marchelli for the several cocktails, dinners, lunches, and basically my entire social life during my last few months of writing. He is always extremely supporting about every aspect of my life and ready to cheer me up with his weird sense of humor.

I would like to spend a few words to thank my friends outside Bonn, who never let me down despite the distance. Again, they are too many to list them all here, but I would like to especially thank Stefano Casti, who has been my best friend since we were 14. He never stopped caring for me, in my best and worst times and for this I will always be grateful to him. I also want to thank Steve Heim for always giving me good advice and for the great trips together. I am sure our friendship will survive no matter where we will end up in the world.

Last, but not least, I would like to thank my family. I have no words to describe their unconditional love and support, wherever I am in the world and whatever I do. They have been my first teachers and they always push me to give my best no matter what I do. Without them, I would not have lived this amazing life.

To all the people I mentioned here and to all my other friends, you made me the person I am today and my achievements are also yours. Thank you.

The work presented in this thesis is partially supported by the DFG through the Mapping on Demand project, grant number FOR 1505: Mapping on Demand and through German's Excellence Strategy EXC-2070-390732324: PhenoRob. The financial support of the DFG is gratefully acknowledged.

Contents

Chapter 1

Introduction

1.1 Mapping the Environment with Robots

Building a 3D model of the environment is important for many applications. It allows for inspecting a building, monitoring a construction site, planning a patrolling path for automated security robots, etc. However, the process that leads to such a model is generally relatively slow and expensive, since it often requires trained personnel operating professional equipment, such as calibrated cameras or terrestrial laser scanners. Moreover, for certain applications it is not possible to map the environment using the traditional techniques. For instance, in a rescue scenario it might be too dangerous to send trained operators for scanning the environment, although having a 3D model could greatly help planning the rescue operations. Another example is augmented reality, which requires the 3D model of the environment for a realistic interaction between virtual objects and real world. In this case, the user is not trained and does not have the equipment for building a 3D model beforehand.

With the advent of inexpensive sensors and robotic platforms, an alternative approach for mapping the environment in 3D is the use of mobile robots. Consider the already mentioned rescue scenario. A robot could be deployed for exploring the dangerous environment and identify risks or people in danger before a team of humans takes action. We distinguish two main challenges for achieving such a task. First, the robot has to be able to reconstruct in 3D the environment online, i.e., it has to incrementally update the map as it receives information from its sensors. Second, the robot must autonomously navigate the environment to efficiently acquire information from the sensors.

Typically, when solving these challenges, the general assumption is that the environment is static, i.e., it does not change over time. However, in the real world, this is usually not the case. Consider for example a city, where old buildings are continuously demolished and new buildings are erected. The 3D model of such cities needs to be constantly updated to reflect the real state of the world. A different example is a room populated with people. In this case, the people are not part of the room to be mapped and need to be discarded when building the model. The two mentioned examples illustrate two different types of non-static environments. In the first case, the environment changes in the long term, and such changes need to be integrated in a previously existing model. In contrast, in the second case, the environment contains dynamic elements during the mapping process. These elements interfere with the process and must not be included in the model.

In this thesis, we address the problem of active 3D reconstruction with robots, and we focus in particular on the four above mentioned problems. Specifically, we first present a solution for creating a model in real-time using a cheap, consumer sensor. Second, we address the problem of selecting the best viewpoints in the environment to maximize the information obtained from the sensors. Third, we deal with the long term changes of the environment, that should be added in the model. Finally, we tackle the problem of discarding dynamic elements that interfere with the mapping process. All the presented algorithms are able to operate online on a robotic platform, and are tested on real-world data. Moreover, all the methods presented in this thesis have been published at peer-reviewed international workshops, conferences, and journals. Furthermore, some of them have been made available as open source software. Finally, during our work we recorded new datasets for testing our approaches, which we released publicly and provided a benchmark for evaluating future work.

This thesis is organized as follows. We first provide, in Chapter 2, the basic knowledge necessary to better understand the thesis. In Part I, we address the problem of autonomously reconstructing static environments in 3D using a robot. In particular, in Chapter 3 we focus on reconstructing dense 3D models using a consumer RGB-D sensor, while in Chapter 4 we present an exploration algorithm that selects the best viewpoints to reconstruct the environment. In Part II, we address the two aforementioned cases of non-static environment. Specifically, in Chapter 5, we propose an algorithm that detects the changes of the environment w.r.t. an existing model, in order to update it, while in Chapter 6, we propose a technique to detect dynamic elements of the environment during the reconstruction process, with the purpose of filtering them out from the sensor information.

1.2 Main Contributions

In this thesis, we address several challenges that arise when performing autonomous reconstruction with robots. We propose a novel approach for each of the mentioned problems. The first contribution of this work is a novel approach for simultaneous localization and mapping with an RGB-D sensor. Our approach allows to incrementally build a dense mesh of the environment, while it accurately tracks the pose of the sensor. Our technique is based on the work of Canelhas *et al.* [16], in combination with voxel hashing [81] to efficiently store the model of the environment and enable the reconstruction of larger scenes. In addition, we propose a novel technique to exploit RGB information to make the algorithm robust in situations where only poor structural information is available. Finally, our implementation exploits the GPU to achieve real-time performance. The approach has been tested on a popular benchmark and its performance surpasses the state of the art in situations with low structural information. The method is explained in-depth in Chapter 3.

The second contribution of this thesis is a novel approach for autonomous exploration, which targets specifically micro aerial vehicles [87]. Our method selects iteratively the next viewpoint that provides the best information gain for the sensor. The information gain computation is based on the sensor model proposed by Pizzoli *et al.* [91]. In addition, we take into account the cost of reaching a new viewpoint in terms of distance and predictability of the flight path for a human observer. Finally, our approach additionally selects paths that reduce the risk of crashes when the expected battery life comes to an end, while still maximizing the information gain in the process. We thoroughly tested our approach and the experiments show that it offers an improved performance compared to other state-of-the-art algorithms in terms of precision of the reconstruction, execution time, and smoothness of the path. The method is described in-depth in Chapter 4.

The third contribution of this thesis is a change detection algorithm that operates on a short sequence of images to detect changes in a 3D model [88]. Our approach finds inconsistencies between pairs of images by re-projecting an image onto another one by passing through the given 3D model. This process leads to ambiguities, which we resolve by combining multiple images such that the 3D location of the change can be estimated. A focus of our approach is that it can be executed fast enough to allow the operation on a mobile system. We tested it on existing datasets as well as on our own image sequences and 3D models, which we publicly shared. Our experiments show that our method quickly finds changes in the geometry of a scene. Moreover, we released the implementation of our approach as open source code. Further details are described in Chapter 5.

The fourth contribution of this thesis is a technique to detect and reject dynamic elements in the environment that builds on top of our RGB-D SLAM algorithm [84]. For detecting dynamics, we exploit the residuals obtained after an initial registration together with the explicit modeling of free space in the model. We evaluated our approach on existing datasets and provide a new dataset consisting in highly dynamic scenes. We publicly shared our dataset, together with the ground truth for both the trajectory of the RGB-D sensor, obtained by a motion capture system, and the model of the static environment, built using a high-precision terrestrial laser scanner. Our experiments show that our approach often surpasses other state-of-the-art dense SLAM methods, in terms of both tracking accuracy and model accuracy. Moreover, we released the implementation of our approach as open source code. The method is described in-depth in Chapter 6.

In sum, this thesis presents four contributions, that target different challenges in the context of active 3D environment reconstruction for mobile robots. All our approaches are designed to work online on a robot, and have been tested on real-world data, showing improved performance compared to the state of the art. In addition, we publicly released two new datasets and two open-source libraries in the context of change detection and 3D reconstruction of dynamic environments.

1.3 Publications

Parts of this thesis have been published in the following *peer-reviewed* workshop, conference and journal articles:

- E. Palazzolo and C. Stachniss. Change Detection in 3D Models Based on Camera Images. In *9th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017

- E. Palazzolo and C. Stachniss. Information-Driven Autonomous Exploration for a Vision-Based MAV. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W3:59–66, 2017

- E. Palazzolo and C. Stachniss. Effective Exploration for MAVs Based on the Expected Information Gain. *Drones*, 2(1), 2018

- E. Palazzolo and C. Stachniss. Fast Image-Based Geometric Change Detection Given a 3D Model. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018

- E. Palazzolo, J. Behley, P. Lottes, P. Giguère, and C. Stachniss. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019

Chapter 2

Basic techniques

2.1 Least-Squares

To reconstruct in 3D the environment online, and localize the used sensor in the model, we present in this thesis a simultaneous localization and mapping technique, in Chapter 3 and Chapter 6. Such techniques are often formulated as a minimization problem solved using the non-linear least-squares method. The least-squares method is an approach that minimizes the sum of squared errors to approximate the solution of an overdetermined system. In this section, we provide a short overview of the non-linear least-squares formulation used in the context of robot mapping and we describe the most common algorithms to find a solution.

2.1.1 Non-Linear Least-Squares Problem

Given a vector of values $\mathbf{x} = \left[\mathbf{x}_1^\top, \ldots, \mathbf{x}_n^\top\right]^\top$ and a set of functions $f_i(\mathbf{x})$, with $i = 1, \ldots, m$, the problem consists in minimizing the cost fuction $E(\mathbf{x})$ defined as:

$$E(\mathbf{x}) = \sum_{i=1}^{m} (f_i(\mathbf{x}))^2, \tag{2.1}$$

where $m \geq n$. In particular, we want to find the value $\mathbf{x}^*$ such that:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} E(\mathbf{x}). \tag{2.2}$$

In this thesis, $f_i(\mathbf{x})$ will always represent errors between estimated values $\boldsymbol{h}_i(\mathbf{x})$ and measured ones $\mathbf{z}_i$. We define such error as:

$$\boldsymbol{e}_i(\mathbf{x}, \mathbf{z}_i) = \boldsymbol{h}_i(\mathbf{x}) - \mathbf{z}_i, \tag{2.3}$$

where $\boldsymbol{h}_i(\mathbf{x})$ maps the state $\mathbf{x}$ to a predicted measurement and is, in general, non-linear. Therefore, linearizations are needed in practice. From Equation (2.3), we define the *residuals* as the individual squared error terms:

$$r_i = e_i(\mathbf{x}, \mathbf{z}_i) = \boldsymbol{e}_i(\mathbf{x}, \mathbf{z}_i)^\top \boldsymbol{\Lambda}_i \boldsymbol{e}_i(\mathbf{x}, \mathbf{z}_i), \tag{2.4}$$

where $\boldsymbol{\Lambda}_i$ is the information matrix, also called weights matrix. Thus, Equation (2.1) is rewritten as:

$$E(\mathbf{x}) = \sum_{i=1}^{m} e_i(\mathbf{x}, \mathbf{z}_i). \tag{2.5}$$

2.1.2 Gauss-Newton Algorithm

The Gauss-Newton algorithm is a numerical approach to solve non-linear least squares problems. The idea is to approximate the residuals by their Taylor expansion, in the neighborhood of a linearization point $\breve{\mathbf{x}}$. Therefore, the error in Equation (2.3) is espressed w.r.t. $\breve{\mathbf{x}}$, plus an update vector $\Delta\mathbf{x}$:

$$\boldsymbol{e}_i(\breve{\mathbf{x}} + \Delta\mathbf{x}, \mathbf{z}_i) \approx \boldsymbol{e}_i(\breve{\mathbf{x}}, \mathbf{z}_i) + \mathbf{J}_i \Delta\mathbf{x} = \breve{\boldsymbol{e}}_i + \mathbf{J}_i \Delta\mathbf{x}, \tag{2.6}$$

where $\mathbf{J}_i$ is the Jacobian:

$$\mathbf{J}_i = \left. \frac{\partial \boldsymbol{e}_i(\mathbf{x}, \mathbf{z}_i)}{\partial \mathbf{x}} \right|_{\mathbf{x} = \breve{\mathbf{x}}}. \tag{2.7}$$

Given this approximation, we can write the linearized version of Equation (2.5) as:

$$\begin{aligned}
E(\breve{\mathbf{x}} + \Delta\mathbf{x}) &= \sum_{i=1}^{m} e_i(\breve{\mathbf{x}} + \Delta\mathbf{x}, \mathbf{z}_i) & (2.8)\\
&= \sum_{i=1}^{m} \boldsymbol{e}_i(\breve{\mathbf{x}} + \Delta\mathbf{x}, \mathbf{z}_i) \boldsymbol{\Lambda}_i \boldsymbol{e}_i(\breve{\mathbf{x}} + \Delta\mathbf{x}, \mathbf{z}_i) & (2.9)\\
&\approx \sum_{i=1}^{m} (\breve{\boldsymbol{e}}_i + \mathbf{J}_i \Delta\mathbf{x})^\top \boldsymbol{\Lambda}_i (\breve{\boldsymbol{e}}_i + \mathbf{J}_i \Delta\mathbf{x}) & (2.10)\\
&= \sum_{i=1}^{m} \Delta\mathbf{x}^\top \mathbf{J}_i^\top \boldsymbol{\Lambda}_i \mathbf{J}_i \Delta\mathbf{x} + 2\mathbf{J}_i^\top \boldsymbol{\Lambda}_i \breve{\boldsymbol{e}}_i \Delta\mathbf{x} + \breve{\boldsymbol{e}}_i^\top \boldsymbol{\Lambda}_i \breve{\boldsymbol{e}}_i & (2.11)\\
&= \sum_{i=1}^{m} \Delta\mathbf{x}^\top \mathbf{H}_i \Delta\mathbf{x} + 2\mathbf{b}_i \Delta\mathbf{x} + c_i & (2.12)\\
&= \Delta\mathbf{x}^\top \left(\sum_{i=1}^{m} \mathbf{H}_i \right) \Delta\mathbf{x} + 2 \left(\sum_{i=1}^{m} \mathbf{b}_i \right) \Delta\mathbf{x} + \sum_{i=1}^{m} c_i & (2.13)\\
&= \Delta\mathbf{x}^\top \mathbf{H} \Delta\mathbf{x} + 2\mathbf{b} \Delta\mathbf{x} + c. & (2.14)
\end{aligned}$$

Since Equation (2.14) is in quadratic form, we can obtain the increment $\Delta\mathbf{x}^*$ that, applied to the current guess, gives a better solution. Specifically, $\Delta\mathbf{x}^*$ is computed by solving the linear system:

$$\mathbf{H} \Delta\mathbf{x}^* = -\mathbf{b}. \tag{2.15}$$

Note that, in practice, this system is often solved using sparse Cholesky factorization, since $\mathbf{H}$ is a symmetric positive semidefinite matrix. In this way, we avoid the computationally expensive operation of inverting $\mathbf{H}$. The solution is optimal for the linearized approximation of Equation (2.5). However, since such equation is non-linear, we need to apply the increment and iterate the procedure until no substantial improvement is made.

2.1.3 Levenberg–Marquardt Algorithm

The Gauss-Newton algorithm has the problem that the computed increment $\Delta\mathbf{x}^*$ can overshoot the optimal estimate, leading to an increase of Equation (2.5), i.e., taking it further away from the minimum. The Levenberg-Marquardt algorithm tries reduce this problem by solving a dampened version of Equation (2.15):

$$(\mathbf{H} + \lambda\mathbf{I})\Delta\mathbf{x}^* = -\mathbf{b}, \tag{2.16}$$

where $\mathbf{I}$ is the identity matrix, and λ is the dampening factor, which is dynamically updated at every iteration.

2.1.4 Huber Estimator

A common problem of the least squares formulation is the fact that it is sensitive to outliers. In particular, as Equation (2.5) is quadratic, outliers results in large residuals and strongly influence the optimization process. Therefore, a possible solution is to use a robustified version of Equation (2.5):

$$E(\mathbf{x}) = \sum_{i=1}^{m} \rho(e_i(\mathbf{x}, \mathbf{z}_i)), \tag{2.17}$$

where $\rho(e)$ is a robust kernel. A common choice of $\rho(e)$ is the Huber kernel [40]:

$$\rho(e) = \begin{cases} e^2 & e < k \\ 2k|e| - k^2 & \text{otherwise} \end{cases}, \tag{2.18}$$

where k is a small constant. In this way, the error terms close to zero will be scaled quadratically, while the large values will be scaled linearly.

2.2 Information Theory

The focus of this thesis is the autonomous 3D reconstruction of the environment. To achieve full autonomy, a fundamental component is the exploration algorithm described in Chapter 4. This kind of algorithms is based on the information theory [71]. Information theory is the mathematical study of coding of information, and it is based on several measures that are important in robotics algorithms. In this section, we summarize the definitions that are useful for understanding exploration algorithms in robotics.

2.2.1 Entropy

The entropy is the average rate at which information is produced by a stochastic source of data. Given a random variable X with possible values $\{x_1, \ldots, x_n\}$, we define its entropy $H(X)$ as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \ln p(x_i), \tag{2.19}$$

where $p(x_i)$ is the probability that $X = x_i$. Given a second random variable Y, we can define the amount of information needed to describe the outcome of X, given that the value of Y is known, as the conditional entropy $H(X \mid Y)$:

$$H(X \mid Y) = \sum_{j=1}^{m} p(y_j) H(X \mid Y = y_j), \tag{2.20}$$

where:

$$H(X \mid Y = y_j) = -\sum_{i=1}^{n} p(x_i \mid y_j) \ln p(x_i \mid y_j). \tag{2.21}$$

The aforementioned definitions assume the random variables taking discrete values. In case X and Y are continuous random variables, it is necessary to replace the sum with an integral over all the possible values of X or Y. Thus, we define the entropy of a continuous random variable X as:

$$H(X) = -\int p(x) \ln p(x) \, dx, \tag{2.22}$$

and the conditional entropy as:

$$H(X \mid Y) = \int p(y) H(X \mid Y = y) dy, \tag{2.23}$$

with x and y values of X and Y, respectively.

2.2.2 Information Gain

The most important measure for exploration algorithms is the information gain. This quantity is the amount of information obtained about one random variable through observing another random variable. The information gain can be defined using the entropy and the conditional entropy as:

$$I(X,Y) = H(X) - H(X \mid Y). \tag{2.24}$$

More explicitly, the information gain is defined as:

$$I(X,Y) = \sum_{j=1}^{m} \sum_{i=1}^{n} p(x_i, y_j) \ln \left(\frac{p(x_i, y_j)}{p(x_i)p(y_j)} \right), \tag{2.25}$$

or, in the continuous case:

$$I(X,Y) = \iint p(x,y) \ln \left(\frac{p(x,y)}{p(x)p(y)} \right) dx\, dy. \tag{2.26}$$

Figure 2.1: Example of RGB-D sensors. From left to right: Microsoft Kinect, Asus Xtion Pro Live, Intel RealSense D435. Courtesy of Microsoft, Asus and Intel.

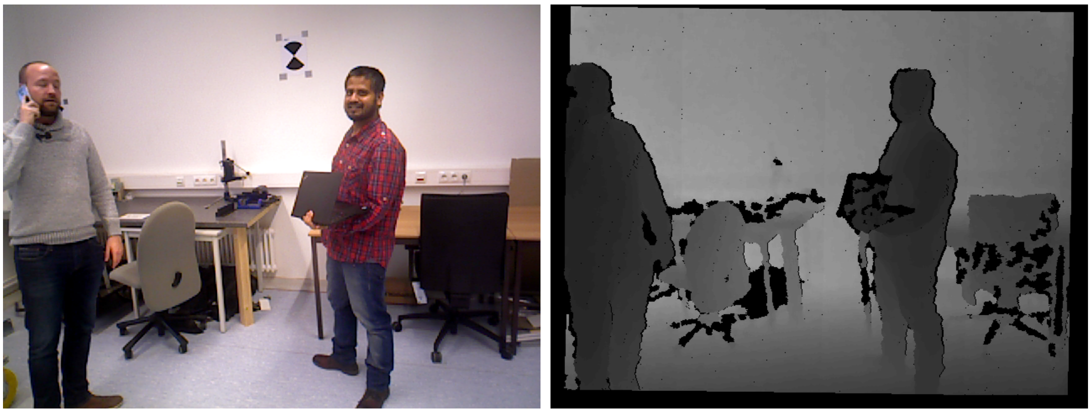


Figure 2.2: Output images of RGB-D sensors. Left: RGB image. Right: Depth image. The brighter a pixel is, the farther away it is from the sensor.

2.3 RGB-D Sensors

In this thesis, to perform online 3D reconstruction of the environment, we rely on RGB-D SLAM algorithms, see Chapter 3 and Chapter 6. The input of these algorithms is not only a sequence of RGB images, but also the 3D position of each pixel w.r.t. the sensor. This is possible thanks to RGB-D sensors. This kind of sensor has become very popular in computer vision and robotics since Microsoft introduced its Kinect in 2010 as an interaction device for the gaming console Xbox 360. Since then, many cheap RGB-D sensors became available on the market, either based on structured light or on time of flight [106]. Figure 2.1 shows some examples of RGB-D sensors, such as the Asus Xtion Pro Live or the Intel RealSense D435.

The output of these sensors is depicted in Figure 2.2. The left image in the figure is an RGB image, i.e., each pixel stores the color of a point in the real world. The right image is a depth image, i.e., each pixel stores the Z coordinate of that point w.r.t. the reference frame of the sensor, conventionally the one shown in Figure 2.3. Given the depth image, and knowing the intrinsic parameters of the camera, we can compute, for every pixel $\mathbf{p} = \begin{bmatrix} u & v \end{bmatrix}^\top$, its 3D position $\mathbf{x}$:

$$\mathbf{x} \quad = \quad \begin{bmatrix} \frac{u - c_x}{f_x} D(\mathbf{p}) \\ \frac{v - c_y}{f_y} D(\mathbf{p}) \\ D(\mathbf{p}) \end{bmatrix}, \tag{2.27}$$

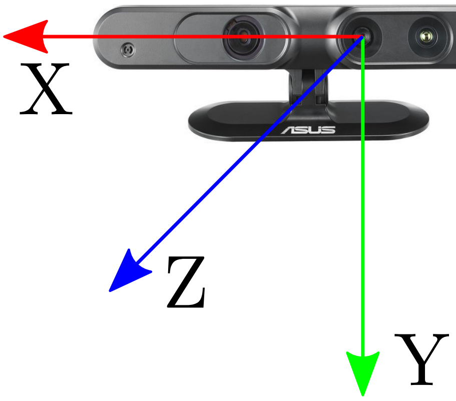


Figure 2.3: Reference frame of an RGB-D sensor. X is pointing right, Y is pointing down and Z is pointing forward w.r.t. the camera.

where $D(\mathbf{p}) : \mathbb{R}^2 \rightarrow \mathbb{R}$ is a function that maps a pixel to its depth, c_x and c_y are the coordinates of the principal point of the camera, f_x and f_y are the focal lenghts in terms of pixels of the camera.

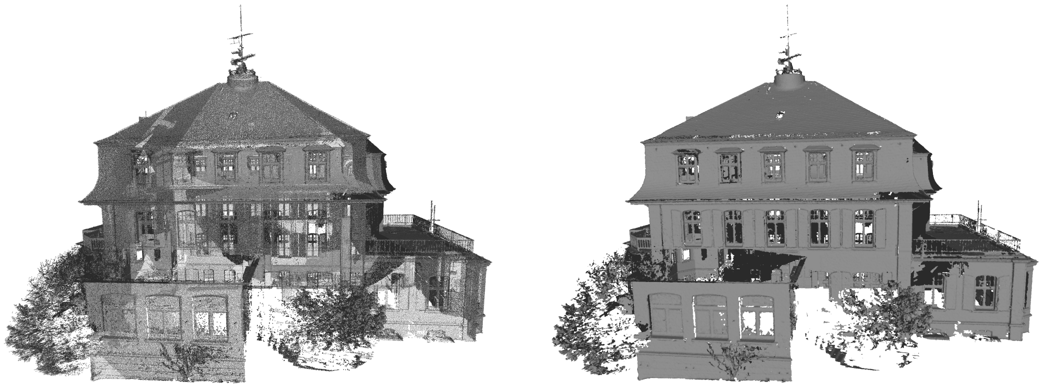


Figure 2.4: Left: an example of point cloud. Right: an example of mesh.

2.4 3D Map Representation

For every task this thesis focuses on, it is necessary to store efficiently the 3D map of the environment. There are different techniques to achieve that, according to the application. Common representations for 3D models are point clouds, which are lists of all the points of the model, or polygon meshes, which are a collection of vertices, edges and faces that describe the shape of the model, see Figure 2.4 for an example. We use the latter representation in Chapter 5. However, such representations generally require a large amount of memory for storage. Moreover, they describe exclusively the *shape* of the environment, and it is hard to encode additional information that might be required. Therefore, for online applications, it is necessary to use a different representation. Specifically, the 3D space is usually discretized and the map is represented in a 3D grid, where each cell, or *voxel*, stores some information about the portion of space that contains. This information could be for example whether the space is occupied or not, the average cordinates of the points in that space, etc. In this thesis, we use such discrete representation in Chapter 3, Chapter 4 and Chapter 6. In this section, we describe the two techniques that we use to efficiently represent the map of the environment in this thesis.

2.4.1 Truncated Sign Distance Function

Typical model representations only store information about the free and occupied space. However, knowing the *distance* from the closest point in space that is occupied can be useful in many applications, such as localization, mapping, obstacle avoidance, etc. To this end, it is useful to represent the 3D model using truncated signed distance function, or TSDF, as originally proposed by Curless and Levoy [25]. The idea is to represent the world with a 3D voxel grid in which each voxel contains an SDF value. The SDF is a function $V_{\mathrm{SDF}}(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}$ that

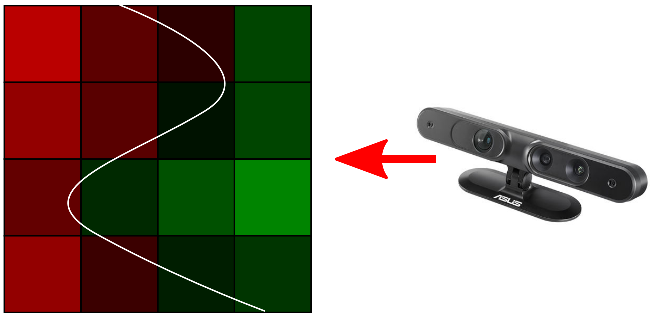


Figure 2.5: A 2D example of TSDF. Darker cells are closer to the surface. Green cells are in front of the surface. Red cells are behind the surface.

returns, given a point in space, its distance to the nearest surface. This distance has a positive sign if the point is in front of the surface and a negative sign if the point is behind it. In this way, the surface is implicitly represented by the set of points for which $V_{\mathrm{SDF}}(\mathbf{x}) = 0$. In practice, the SDF values are truncated to a given maximum value. This is because the SDF is computed from a sensor with a projective model, thus the computed distances are more inaccurate the farther from the surface they are. Moreover, it is impossible to accurately compute the negative side of the SDF since it is occluded by the surface itself. Voxels with an SDF outside the truncation region are commonly just set to the maximum value. Figure 2.5 shows an example of SDF in 2D.

Additionally, each voxel contains a weight w, that represents how reliable the SDF value is at that location. This leads to an improved robustness to outliers. The most common weighting functions are uniform weighting and linear weighting [25], but alternative functions have also been proposed (e.g. exponential weighting [15]). Thus, the existing model is updated using a running weighted average:

$$V_{\mathrm{SDF}}(\mathbf{x})_{t+1} = \frac{V_{\mathrm{SDF}}(\mathbf{x})_t w(\mathbf{x})_t + V_{\mathrm{SDF}}(\mathbf{x}) w(\mathbf{x})}{w(\mathbf{x})_t + w(\mathbf{x})}, \tag{2.28}$$

$$w(\mathbf{x})_{t+1} = \min(w(\mathbf{x})_t + w(\mathbf{x}), w_{\max}), \tag{2.29}$$

where $w_{\max}$ is a maximum weight, that increases the robustness to possible changing elements in the environment.

To enable the direct use of color information in the camera tracking, and to enable texturing of the mesh, it is possible to store, for each voxel, the color obtained by projecting it onto the RGB image. The colors are updated among multiple scan using the same running average as in Equation (2.28):

$$R(\mathbf{x})_{t+1} = \frac{R(\mathbf{x})_t w(\mathbf{x})_t + R(\mathbf{x}) w(\mathbf{x})}{w(\mathbf{x})_t + w(\mathbf{x})}, \tag{2.30}$$

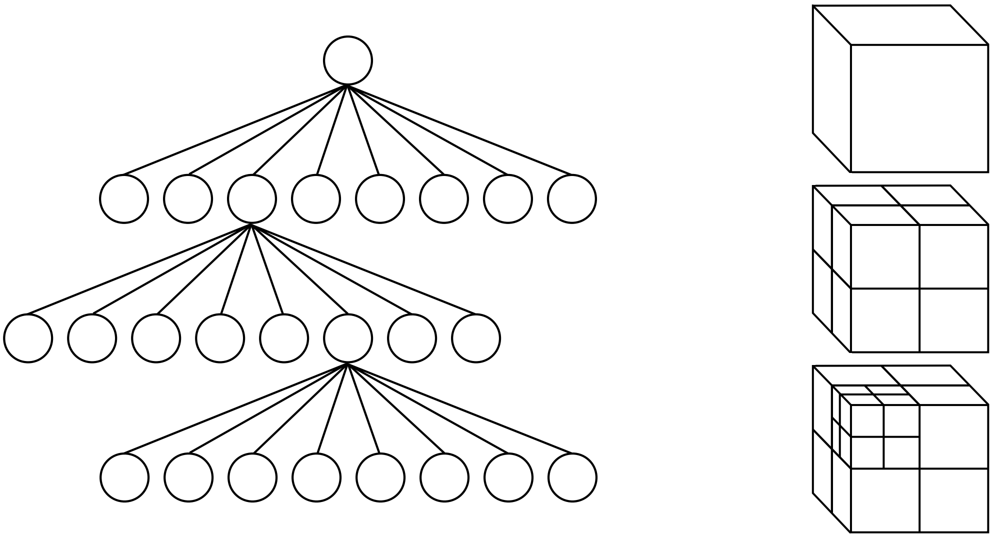


Figure 2.6: Left: an example of octree. Right: space subdivision in octants.

where $R(\mathbf{x})$ represents the red value of the color. The green and blue values $G(\mathbf{x})$ and $B(\mathbf{x})$ are computed in an identical way.

The TSDF representation is heavily used in dense RGB-D mapping approaches, such as the ones described in Chapter 3 and Chapter 6. The surface can be extracted from this representation in a highly parallelizable way using the marching cubes algorithm [70], usually implemented on the GPU.

2.4.2 Compressing a Voxel Grid using Octrees

In the introduction of Section 2.4, we mentioned how representing the space in a voxel grid can be useful for online applications. However, representing the grid as a full, multi-dimensional array of voxels has its limitations. In particular, the voxel grid has the shape of a rectangular cuboid, and contains a number of voxel n computed as:

$$n = \frac{lwh}{s}, \tag{2.31}$$

where l, w and h are, respectively, the length, width and height of the cuboid, and s is the voxel size, i.e., the length of one side of a voxel. Equation (2.31) shows that increasing s is the only way of reducing the memory occupied by a large volume, but that leads to a less detailed model. Moreover, if the mapped area has not the shape of a cuboid, the voxel grid will contain a large number of unused voxels. Therefore, for many applications, such as the one described in Chapter 4, it is necessary to compress the grid.

To this end, one technique that has proved to be very efficient is the use of octrees [74]. An octree is a tree data structure in which each node has exactly eight children, as depicted on the left side of Figure 2.6. By using an octree, it is

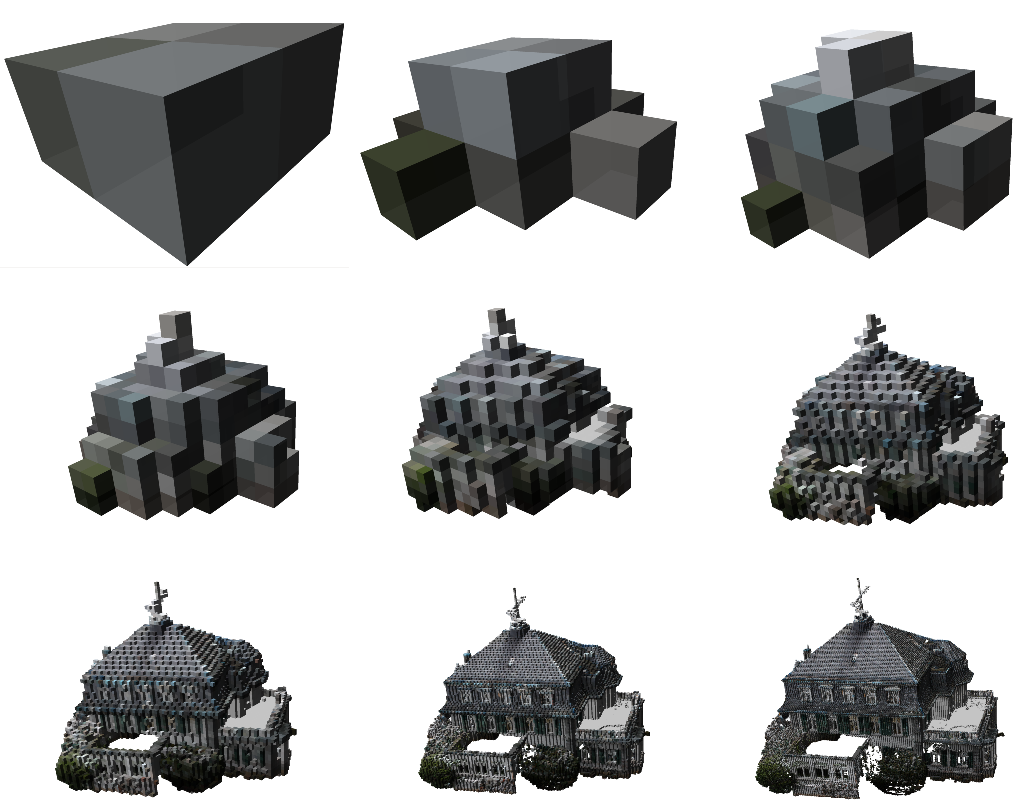


Figure 2.7: Coarse to fine visualization of an octree-based model.

possible to partition a 3D space into eight octants. Following this principle, each node represents a protion of space, and its children represent the eight octants of that portion, i.e., eight smaller portions of space, see right side of Figure 2.6. In this way, the model is represented in a coarse-to-fine fashion, in which similar voxels are compressed in a single node, while volumes containing more details are represented by a larger number of smaller voxels. Figure 2.7 shows the different levels of an octree-based model. Each picture shows a deeper level of the octree compared to the previous one, i.e., in each successive picture the voxels are eight times smaller.

Part I

Static Environments

Chapter 3

Simultaneous Localization and Mapping with RGB-D Sensors

Mapping and localization are essential capabilities of robotic systems operating in real-world environments. This task is usually solved online in an alternating fashion, where one determines the pose w.r.t. the map built so far and then use the estimated pose to update the map. Sensors that provide dense depth information such as an RGB-D camera or a stereo camera, enable the creation of a map dense enough to provide a 3D model of the environment in real time, during the SLAM process. For such a purpose, a common approach is to represent the environment in the form of a TSDF. As explained in Section 2.4.1, such representation allows to efficiently extract a 3D mesh of the environment when needed. In this way, it is possible to deploy a robot that scans the environment with its sensor and builds, in real time, the desired 3D reconstruction.

In this chapter, we present an approach for the tracking of an RGB-D sensor, while reconstructing in 3D the environment. In particular, we focus on the efficiency of the algorithm, both in terms of memory consumption and speed. Moreover, our technique exploits RGB information to be robust in environments with low structural information.

3.1 Efficient RGB-D SLAM

The main contribution of this chapter is an efficient SLAM algorithm based on a TSDF and inspired by the work of Canelhas *et al.* [16] combined with voxel hashing [81]. It provides a greatly reduced memory consumption and thus allows for mapping of larger areas, while being able to maintain real-time performance by exploiting the GPU. In addition, we integrate color directly into a voxelized representation of the TSDF allowing us to use color for tracking without an explicit projection or raycasting.

Extensive evaluations on the TUM RGB-D dataset [125] show the versatility and robustness of our approach reaching in most settings better performance than the original approach by Canelhas *et al.* [16]. In addition, we show that our improvements lead to better results in situations with low geometric detail and in situations where we want to map larger areas.

In sum, we make three key claims: our RGB-D mapping approach is (i) robust to scenes that contain low structural information, as long as the texture information is enough, (ii) able to allocate voxels only when necessary, leading to a reduced memory consumption, and (iii) able to achieve real-time performance, i.e., it runs faster than the framerate of an RGB-D sensor, which typically is 30 Hz.

3.1.1 Voxel Hashing for Efficient Storage

Storing the TSDF in a 3D grid as described in Section 2.4.1 is common in RGB-D mapping approaches. However, to reduce the memory consumption and enable reconstructions of larger scenes, we do not store the SDF in a real grid. Instead, we only allocate the voxels within the truncation region, i.e., around obstacles, and index them using a spatial hashing function, in a similar way as the one proposed by Nießner *et al.* [81].

The idea is to represent the world sparsely by only allocating small blocks of voxels (in our implementation $8 \times 8 \times 8$ voxels) inside the truncation region of the TSDF. These blocks are serially stored in an array and indexed using a hash table. Each entry of the hash table stores the pointer to the voxel block array and the 3D location of the block. To deal with collisions of the hash function, the hash entries are organized into buckets, hence the need of the 3D location of the block stored into the entry. To access a voxel block, we first compute the hash function to identify the location of the bucket containing the block. For this purpose we use the commonly used hash function (e.g., [132], [81], [59]):

$$h(x, y, z) = (p_1 x \oplus p_2 y \oplus p_3 z) \bmod M, \tag{3.1}$$

where $p_1 = 73856093$, $p_2 = 19349669$, $p_3 = 83492791$, M is the number of buckets, $\oplus$ is the logical XOR operator, and mod is the modulo operator.

Using the result of Equation (3.1), we access a bucket of entries and check for each entry in the bucket if the 3D location of the block corresponds to the desired one. Once we locate such entry, we access the voxel block in the array, using the stored pointer. Figure 3.1 illustrates the described data structure.

In our implementation, we fuse an RGB-D scan given its pose by first allocating all the voxel blocks within the camera frustum and the truncation region,

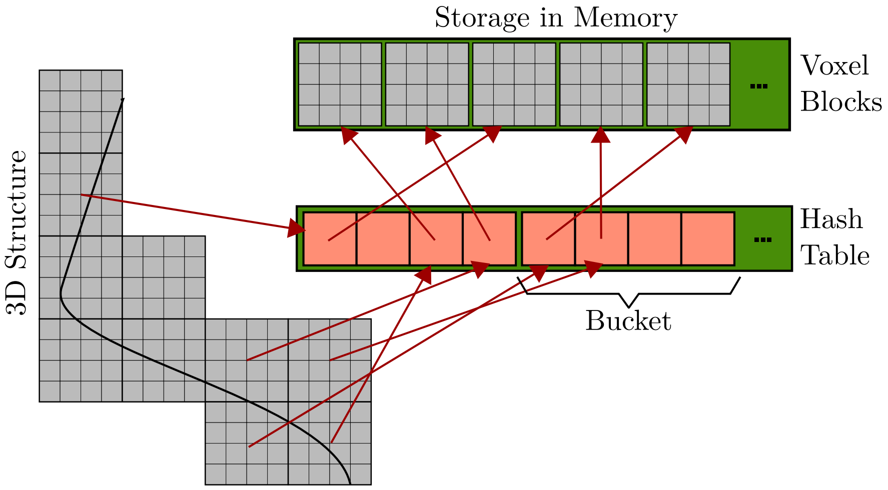


Figure 3.1: Example of how the data structure used for voxel hashing works. Instead of storing a full grid, we only store voxel blocks that are addressed by a hash table.

identified by performing a ray casting from the camera pose. Then, for each allocated voxel inside the camera frustum, we project its center onto the RGB and depth images and update respectively its color and its SDF value as described in Section 2.4.1.

3.1.2 Pose Estimation

For estimating the pose of the sensor, we use a point-to-implicit approach such as the ones by Canelhas *et al.* [16] and Bylow *et al.* [15]. In contrast to KinectFusion [79] and similar methods, our approach does not generate synthetic images from the model. Instead, we use an alternative technique and directly align a point cloud to the SDF. The reason for that is that the SDF provides by definition the distance of a point to the closest surface, i.e., it is possible to use the SDF value directly as an error function, as we explain in the rest of this section. In addition to the existing point-to-implicit techniques, we exploit the color information contained in the model to improve the alignment.

Each frame of an RGB-D sensor consists into a depth image and a color image. Given a pixel $\mathbf{p} = \begin{bmatrix} u & v \end{bmatrix}^\top$, we define the functions $D(\mathbf{p}) : \mathbb{R}^2 \rightarrow \mathbb{R}$ and $I(\mathbf{p}) : \mathbb{R}^2 \rightarrow \mathbb{R}$, which map a pixel respectively to its depth and its intensity. We denote with $\mathbf{x}$ the 3D point resulting from the back-projection of a pixel $\mathbf{p}$, as explained in Section 2.3

We represent a camera pose as a 3D transformation $\mathbf{T} \in \mathbb{SE}(3)$. A small rigid-body motion can be written in minimal form using the Lie algebra representation:

$$\boldsymbol{\xi} \quad = \quad \begin{bmatrix} \Delta\alpha & \Delta\beta & \Delta\gamma & \Delta x & \Delta y & \Delta z \end{bmatrix}^\top . \tag{3.2}$$

The vector $\boldsymbol{\xi} \in \mathfrak{se}_3$ can be converted into the corresponding Lie group $\mathbb{SE}(3)$ by computing $\mathbf{T}' = \exp(\hat{\boldsymbol{\xi}})$, where:

$$\hat{\boldsymbol{\xi}} \quad = \quad \begin{bmatrix} 0 & -\Delta\gamma & \Delta\beta & \Delta x \\ \Delta\gamma & 0 & -\Delta\alpha & \Delta y \\ -\Delta\beta & \Delta\alpha & 0 & \Delta z \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.3}$$

To represent the current model, we use a voxel-based representation where we store, in each voxel, TSDF and color information. We define the functions $V_{\mathrm{SDF}}(\mathbf{x})$ and $V_{\mathrm{I}}(\mathbf{x})$, that return respectively the SDF and the intensity at position $\mathbf{x}$. As our representation is a discrete space, we obtain the SDF and intensity values at non-integer positions using trilinear interpolation.

Similarly to Canelhas *et al.* [16] and Bylow *et al.* [15], we directly exploit the SDF to define the error function, as such function directly represents the distance of a point from the surface. Thus, we define the error function relative to the depth as:

$$E_d(\boldsymbol{\xi}) \quad = \quad \sum_i^N \underbrace{\left\| V_{\mathrm{SDF}}(\exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i) \right\|^2}_{r_i}, \tag{3.4}$$

where N is the number of pixels in the image, and $\mathbf{x}_i, i \in 0 \ldots N$ is the 3D point corresponding to the i-th pixel, computed using Equation (2.27). The value r_i corresponds to the residual for the i-th pixel.

We additionally exploit the color information to improve the alignment. In contrast to most of the state-of-the-art approaches (e.g., [140, 136]), we do not create an artificial image from the model. Instead, we directly operate on the color information stored in the voxels. As we store in each voxel the color of the closest point of the surface, we define the error function relative to the color as the photometric error between the intensity of the pixels of the current image, and the intensity of the corresponding voxels in the model:

$$E_c(\boldsymbol{\xi}) \quad = \quad \sum_i^N ||V_{\mathrm{I}}(\exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i) - I(\mathbf{p}_i)||^2, \tag{3.5}$$

the joint error function is given by:

$$E(\boldsymbol{\xi}) \quad = \quad E_d(\boldsymbol{\xi}) + w_c E_c(\boldsymbol{\xi}), \tag{3.6}$$

with w_c weighting the contribution of the intensity information w.r.t. the depth information.

The goal is to find the camera motion $\boldsymbol{\xi}^*$ that minimizes the error function:

$$\boldsymbol{\xi}^* \quad = \quad \operatorname*{argmin}_{\boldsymbol{\xi}} E(\boldsymbol{\xi}). \tag{3.7}$$

We solve this equation by using the least-squares approximation. We linearize the objective function with a first order Taylor approximation and rewrite the problem in a quadratic form. To do that, we need to compute the Jacobians $\mathbf{J}_{d,i}$ and $\mathbf{J}_{c,i}$ relative to the SDF and the color of point $\mathbf{x}_i$. The Jacobian relative to the SDF is:

$$\mathbf{J}_{d,i} = \frac{\partial V_{\mathrm{SDF}}(\exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i)}{\partial \boldsymbol{\xi}}, \tag{3.8}$$

and using the chain rule, we obtain:

$$\mathbf{J}_{d,i} = \frac{\partial V_{\mathrm{SDF}}(\exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i)}{\partial \exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i} \frac{\partial \exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i}{\partial \boldsymbol{\xi}}. \tag{3.9}$$

The first term of Equation (3.9) is computed numerically from the voxelized model by evaluating the gradient in each of the three directions, with sub-voxel precision using trilinear interpolation. The second term of Equation (3.9) is computed by:

$$\frac{\partial \exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i}{\partial \boldsymbol{\xi}} = \begin{bmatrix} 0 & z_i & -y_i & 1 & 0 & 0 \\ -z_i & 0 & x_i & 0 & 1 & 0 \\ y_i & -x_i & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.10}$$

We compute $\mathbf{J}_{c,i}$ in an analogous way. Having computed the Jacobians, we write the matrix $\mathbf{H}$ and the vector $\mathbf{b}$ as:

$$\mathbf{H} = \mathbf{H}_d + w_c \mathbf{H}_c, \tag{3.11}$$

$$\mathbf{b} = \mathbf{b}_d + w_c \mathbf{b}_c, \tag{3.12}$$

with:

$$\mathbf{H}_d = \sum_i^N \mathbf{J}_{d,i}^\top \mathbf{J}_{d,i}, \tag{3.13}$$

$$\mathbf{H}_c = \sum_i^N \mathbf{J}_{c,i}^\top \mathbf{J}_{c,i}, \tag{3.14}$$

$$\mathbf{b}_d = \sum_i^N \mathbf{J}_{d,i} V_{\mathrm{SDF}}(\exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i), \tag{3.15}$$

$$\mathbf{b}_c = \sum_i^N \mathbf{J}_{c,i}(V_{\mathrm{I}}(\exp(\hat{\boldsymbol{\xi}})\mathbf{T}\mathbf{x}_i) - I(\mathbf{p}_i)). \tag{3.16}$$

Note that for computing $\mathbf{H}_d$ and $\mathbf{b}_d$, we use a Huber estimator to increase the robustness w.r.t. gross errors [16]. Finally, we can compute the increment using Levenberg-Marquardt as:

$$\boldsymbol{\xi}^* = -(\mathbf{H} + \lambda \mathbf{I})^{-1}\mathbf{b}, \tag{3.17}$$

where $\mathbf{I}$ is the identity matrix and λ a regularization term. The new increment is applied to the current transformation and the whole process is iteratively repeated for a given number of iterations or until the norm of the difference between the current increment and the previous one is below a given treshold. As it is typical for this type of approaches, we perform the optimization on three different coarse-to-fine sub-sampling of the input images. Furthermore, we linearly increase λ with the number of iterations, in order to increasingly dampen the increment, the more the solution get close to optimal [16].

3.1.3 Efficient Implementation on the GPU

A focus of our approach is its real-time performance, i.e., the registration and integration of new scans must be faster than the framerate of the sensor, which is typically 30 Hz. To achieve the required performance, we implemented most of the algorithm on the GPU, using the CUDA API by NVIDIA. In particular, the scan fusion algorithm described in Section 3.1.1 consists in two steps, both highly parallelizable. In the first step, we allocate the necessary voxel blocks using ray casting. We project a ray for each pixel of the depth image. Since each pixel is independent from the others, we compute each ray on a different thread on the GPU. The second step is the update of the model. To this end we consider each allocated voxel in the camera frustum using a different thread and update them simultaneously.

The second main block of our approach is the pose registration, see Section 3.1.2. In our implementation, we process every pixel of a new RGB-D image concurrently. For each pixel, we compute the terms of the Jacobians and the relative terms of Equation (3.13) to Equation (3.16) on a different thread on the GPU, summing them together using atomic operations. In our experience, this results more efficient than performing a parallel reduction. We then evaluate Equation (3.17) on the CPU using Cholesky decomposition.

3.2 Experimental Evaluation

The focus of this chapter is a real-time capable TSDF-based mapping approach that is able to operate in environments with little structural information. Our experiments are designed to show the capabilities of our method and to support our key claims, which are: our approach (i) is robust to scenes that contain low 3D structural information, as long as there is texture information available, (ii) is able to allocate voxels only when necessary, leading to a reduced memory consumption, and (iii) is able to achieve real-time performance, i.e., it runs faster than the framerate of an RGB-D camera, which is typically 30 Hz.

Table 3.1: Parameters of our approach in all experiments.

Parameter	Value
Voxel size	0.01 m
Truncation distance τ	0.1 m
Huber constant	0.02
Initial regularization parameter λ	0.002
Weight w_c	0.1

Table 3.2: Absolute Trajectory Error (RMS) [m] on different scenes of TUM dataset. The values illustrate the RMS absolute distance between the estimated and the ground truth trajectory.

Dataset	Ours	SDFT [16]	EF [140]	EF* [140]	MPR [28]
fr1/xyz	0.028	0.020	**0.013**	**0.013**	0.128
fr1/desk	0.097	0.082	0.032	**0.030**	0.106
fr1/floor	0.501	0.675	0.484	**0.469**	0.805
fr3/nst_far	**0.040**	1.317	0.284	0.284	0.894
fr3/nst_near	0.076	1.968	**0.024**	0.027	1.925
fr3/nst_loh	0.152	1.528	**0.034**	0.045	0.445
Max	0.501	1.968	0.484	**0.469**	1.925

We evaluate our approach on the TUM RGB-D dataset [125] and compare it with four other state-of-the-art methods, i.e., SDF Tracker (SDFT) [16], ElasticFusion (EF) [140], and multi-cue photometric point cloud registration (MPR) [28]. In particular we use the open-source implementations of ElasticFusion and MPR, and the code provided by the author of SDF Tracker. We denote with EF* the results of ElasticFusion without loop closures, since no other method in our evaluation closes loops. We run every approach with the respective default parameters provided by the authors. For our approach, we used the parameters listed in Table 3.1.

3.2.1 Performance

The first experiment is designed to show the performance of our approach and to support the claim that it is well suited for scenes that contain low structural geometric information, but texture.

Table 3.2 and Table 3.3 show the absolute trajectory error and the relative pose error for some scenes of the TUM dataset, respectively. The absolute trajectory error values illustrate the RMS absolute distance between the estimated and the ground truth trajectory, while the relative pose error values illustrate the

Table 3.3: Relative Pose Error (RMS) on different scenes of TUM dataset. The values illustrate the RMS drift of the trajectory over one second.

Dataset		Ours	SDFT [16]	EF [140]	EF* [140]	MPR [28]
fr1/xyz	[m/s]	0.024	0.026	**0.018**	**0.018**	0.041
	[deg/s]	1.092	1.538	**0.932**	**0.932**	2.351
fr1/desk	[m/s]	0.059	0.064	0.043	**0.040**	0.061
	[deg/s]	**2.200**	4.044	3.171	2.769	3.333
fr1/floor	[m/s]	0.189	0.241	0.217	**0.170**	0.282
	[deg/s]	7.827	9.512	10.830	**6.708**	10.020
fr3/nst_far	[m/s]	**0.044**	0.299	0.300	0.300	0.249
	[deg/s]	**1.209**	2.552	8.538	8.538	2.746
fr3/nst_near	[m/s]	0.022	0.253	0.015	**0.013**	0.250
	[deg/s]	1.250	7.587	0.758	**0.737**	7.634
fr3/loh	[m/s]	0.027	0.737	0.014	**0.012**	0.026
	[deg/s]	1.428	24.56	0.689	**0.591**	1.276
Max	[m/s]	**0.189**	0.737	0.300	0.300	0.282
	[deg/s]	**7.827**	24.56	10.830	8.538	10.020

RMS drift of the trajectory over one second. In scenes containing structural information, i.e., *fr1/xyz* and *fr1/desk*, our approach is on par with SDF tracker. On the more challenging *fr1/floor* dataset, which contains few geometric cues and scarce texture information, our approach outperforms SDF Tracker and is on par with EF, which also exploits color information to track the sensor. In scenes where structural information is mostly absent our approach is substantially better than SDF Tracker. In particular, on the *fr3/nostructure_texture_far* (*fr3/nst_far*) dataset, ElasticFusion also fails, while our approach is able to correctly track the camera and reconstruct the environment. On the *fr3/long_office_household* (*fr3/loh*) dataset, SDF Tracker fails due to the size of the mapped area, but our approach is able to extend the reconstructed volume as needed by allocating voxel blocks. Finally, the worst performance of our approach is on par with ElasticFusion in terms of absolute trajectory error, and the best in terms of relative pose error, meaning that our approach is robust to failure.

3.2.2 Memory Consumption

The second experiment is to support the claim that our approach is able to allocate voxels only when necessary thanks to the voxel hashing, i.e, our approach

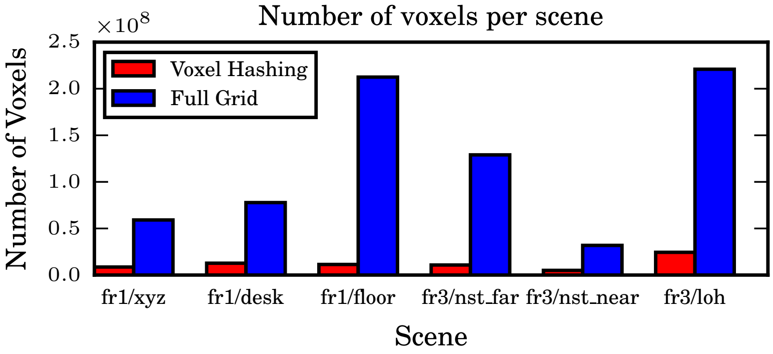


Figure 3.2: Number of voxels allocated to map different scenes using voxel hashing and using a full voxel grid.

requires less memory to map a scene when compared to approaches that use a full voxel grid. Figure 3.2 shows the number of allocated voxels using our approach vs. the number of voxels that would be necessary to allocate using a full voxel grid (without hashing). To compute the number of voxels in the full grid, we first mapped the scene with our approach and then computed the extent of the model along the three axes. From Figure 3.2 it is evident how the use of voxel hashing leads to a significantly smaller memory consumption and thus enable the exploration of larger areas.

3.2.3 Runtime

The last experiment shows that our approach runs fast enough to support online processing of RGB-D data in real-time. We tested our approach on a desktop computer equipped with an Intel Core i7 CPU and an Nvidia GeForce GTX 1080 Ti GPU. Our implementation runs at 46.8 Hz on average on RGB-D images of resolution 320×240. We also tested our approach using VGA resolution images (640×480) and we achieve 22 Hz on average.

3.3 Related Work

With the advent of cheap RGB-D cameras, many approaches for localization and mapping using such sensors were proposed [147]. One classic technique to register point clouds resulting from RGB-D scans is the Iterative Closest Point (ICP) algorithm [11]. More recently Steinbrücker *et al.* [120] proposed an energy minimization algorithm that is two order of magnitude faster than ICP. An alternative approach was proposed by Kerl *et al.* [63]. Their Dense Visual Odometry (DVO) approach registers two consecutive RGB-D frames directly upon each other by minimizing the photometric error. It runs in real-time on a single CPU core and has a small, constant memory footprint. Della Corte *et al.* [28] generalized this approach by introducing support for multiple cues such as depth, color and normal information in a unified way.

A technique that recently became very popular is to represent the map in the form of Truncated Sign Distance Function (TSDF), see Section 2.4.1. The seminal paper of Newcombe *et al.* [79] (KinectFusion) showed the prospects of TSDF-based RGB-D mapping by generating accurate, high detailed maps using only depth information and paved the way for several improvements increasing the versatility and fidelity of RGB-D mapping. Such approach was then extended [56] to be robust to dynamics by using outliers from the registration step to segment out dynamic elements and reconstruct them with a second GPU.

To increase the space that can be mapped, different options were investigated. Whelan *et al.* [139] proposed Kintinuous, an extension of KinectFusion that support large-scale by maintaining the TSDF representation for the currently mapped region and saving the rest of the map as triangular mesh. Chen *et al.* [20], instead, support large-scale by representing the grid using a hierarchical data structure. Steinbrücker *et al.* [119, 121] use an octree instead of a voxel grid, whereas Nießner *et al.* [81] propose to use voxel blocks that can be addressed via a hashing function and these blocks only need to be allocated close to the mapped surface. Thus, it is possible to cover a large area with only few voxels. Kähler *et al.* [60] extend this idea by using a hierarchy of voxel blocks with different resolutions.

Another popular extension of TSDF-based techniques is the use of additional global optimization methods to improve the mapping. Zhou and Koltun [144] employ a typical TSDF frame-to-model registration combined with an offline global graph-based optimization. In addition, they preserve geometric detail of detected densely scanned points of interest in the scene. In a follow-up work [145], they perform frame-to-model registration for small fragments of frames, followed by an offline global non-rigid optimization that allows fragments to deform. In another follow-up work [22] they use a line processes-based offline global optimization to resolve inconsistencies in the model. A more recent approach by Dai *et al.* [27] is BundleFusion. It is a TSDF-based method with voxel hashing for storing the map, but in addition they perform online bundle adjustment in a hierarchical way, i.e., first in a chunk of consecutive frames, then between chunks. Millane *et al.* [76] also use online bundle adjustment, but they represent the map with overlapping TSDF subvolumes localized through feature-based camera tracking. These subvolumes are periodically fused together.

To alleviate the need for raycasting for generating the model image for registration, Canelhas *et al.* [16] and Bylow *et al.* [15] proposed to directly exploit the TSDF for evaluation of the residuals and computation of the jacobians within the error minimization. Slavcheva *et al.* [116], instead, directly minimize the difference between pairs of SDFs, together with global pose optimization and the inclusion of surface normal information. In a follow-up work [115, 117], they enable larger-scale reconstruction by performing the registration over multiple SDF volumes anchored at informative, geometry-rich locations. Finally, a noteworthy approach is the one by McCormac *et al.* [73]. They store per-object TSDFs and perform object-level SLAM by constructing a graph map of reconstructed objects.

Besides using a TSDF, other approaches use other map representations. One example is the use of surfels, which are disks with a normal and a radius. Keller *et al.* [62] use surfels to represent the model of the environment and Whelan *et al.* [140] extend the approach with a deformation graph enabling loop closures.

Most RGB-D approaches use only the depth for mapping and color for visualization, but some [6, 119, 140, 136] also exploit color for estimating the pose of the sensor. Especially in situations with only little geometric information, like flat walls, texture can help to find a correct pose. Our approach uses the direct TSDF evaluation proposed by Canelhas *et al.* [16] and Bylow *et al.* [15]. In addition, we store the model using voxel hashing [81] to substantially reduce the memory consumption and we add the integration of color to enable mapping of areas where depth information is ambiguous.

3.4 Conclusion

In this chapter, we presented a TSDF-based mapping approach able to track the pose of the sensor even in environments with only few structural cues. Our approach tracks the sensor by exploiting directly the TSDF information and the color information encoded in voxels. The TSDF is represented using voxel hashing, in such a way that voxel blocks are only allocated when needed. Furthermore, most computations are parallelized on a GPU, to enable real-time performance. We evaluated our approach on the popular TUM RGB-D dataset and provided comparisons to other state-of-the-art techniques. Our experiments show that our approach leads to an improved pose estimation in situations with low structural information.

Chapter 4

Information-Driven Autonomous Exploration

THE 3D reconstruction approach described in Chapter 3 allows a robot or a human carrying a sensor to easily model the environment by acquiring scans from different points of view. The natural following step is to enable a robot to perform the modeling autonomously, with minimal or no human supervision. When performing *autonomous* 3D reconstruction, the exploration strategy determines the efficiency with which an accurate 3D model of the environment can be obtained. The problem of exploration consists of selecting the best viewpoints to cover the environment with the available sensors to obtain an accurate 3D model. When no a priori information about the environment is available, a popular approach to this problem is the iterative selection of the *next-best-view*. This approach consists in selecting online the next pose for the sensor that best satisfies certain criteria, usually related to the amount of information acquired by the new observations and the cost of actually executing the next action. Since no information about the environment is initially available, this kind of approach works in a greedy fashion, i.e., it is executed online during the mission and considers, at each iteration, the new measurements acquired by the sensor to plan the next pose. Therefore, the algorithm needs to store the information coming from the measurements, as well as which portion of the space has already been explored. Usually, this information is stored in a voxel grid, that is used as a *map* and is updated whenever a new measurement is available. Each voxel contains all the data needed for computing the next-best-view and has three possible states: unknown, free or occupied.

Due to their ability to move freely in all the three dimensions, micro aerial vehicles (MAVs) offer superior mapping capabilities compared to ground vehicles. Specifically, their small size and low weight allow for a high degree of mobility even in challenging environments. However, the use of MAVs introduces sev-

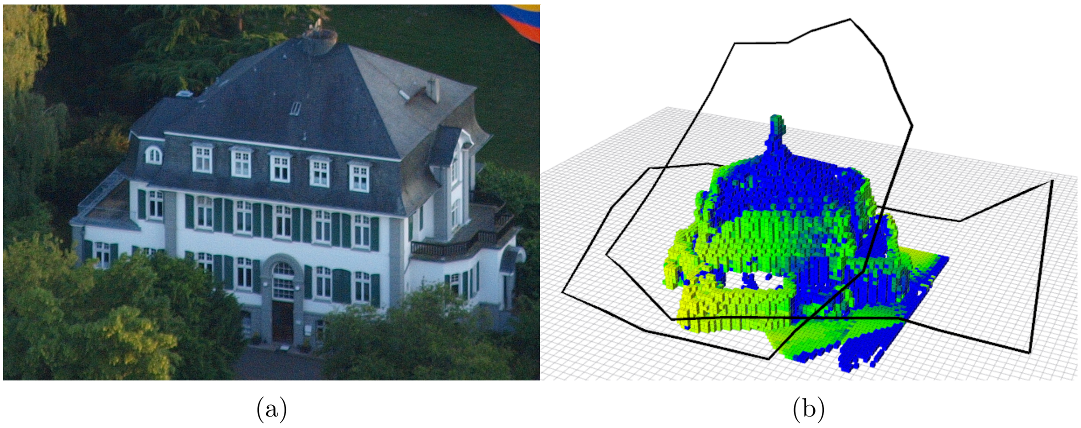


Figure 4.1: Given a scene containing an object of interest (on the left), our algorithm computes the best viewpoints to efficiently map the environment (on the right).

eral challenges that an efficient 3D mapping pipeline must address. The main issue is the reduced payload that this kind of vehicle can carry. This restricts the available sensors mostly to cameras, instead of typically heavier 3D laser scanners. Moreover, the typical flight time of MAVs is limited and, when flying autonomously, the robot has to land safely when the battery life comes to an end. Finally, for outdoor flights, most countries require either manual piloting or a continuous supervision of the MAV by a human operator. In case of autonomous flight monitored by a human, the trajectory of the robot must be predictable and legible [32]. From our experience, an operator can more easily predict the MAV's trajectory if it is free of abrupt changes of direction and thus we prefer exploration paths of such type. In this chapter, we present an autonomous exploration approach for MAVs that takes the above mentioned objectives into account and outputs, during the mission, the next pose to which the robot has to move. It was developed in the context of the DFG research unit Mapping on Demand.

4.1 Autonomous Exploration for MAV

The main contribution of this chapter is an exploration approach that selects in real-time the next-best-view that maximizes the expected information gain of new measurements. In addition, we take into account the cost of reaching a new viewpoint in terms of distance and predictability of the flight path for a human observer. Finally, our approach selects a path that aims to reduce the impact of crashes when the expected battery life comes to an end, while still maximizing the information gain in the process.

We assume the environment to be unknown, but we limit the space of the mission to given boundaries, in the form of a bounding box. We also assume to have in real-time 3D measurements from the sensor, e.g., in the form of a pointcloud or a range image, dense enough w.r.t. the resolution of the voxel map. This data can come for example from a stereo camera, or from multiple observations by a monocular camera. Instead of constraining our approach to a specific sensor, we take into account directly the 3D data, which can come from any existing algorithm or sensor. Moreover, our approach requires the pose of the vehicle to be known. In our implementation, we obtain such information from a VO/IMU/DGPS combination as described in [108], but any other online SLAM system could be used.

Given a bounding box that contains the object of interest, for example a building that should be explored, such as the one in Figure 4.1a, our approach greedily selects, in real time, the next best viewpoint that maximizes a utility function that focuses on:

- reducing the uncertainty in the 3D model,
- producing a flight path with a small number of abrupt turns,
- producing a progressively safer path the longer the MAV is flying to avoid crashes, and
- allowing a safe landing at the starting point within a user-specified time constraint.

We constrain the motion of the MAV on a hull that initially surrounds the given bounding box and then is iteratively refined to always fit the known map at that given point in time. On this hull, we sample the candidate next-best-views and compute an approximation of the expected uncertainty reduction by taking into account the properties of the camera used for reconstruction. Moreover, we take into account a cost function that prevents the MAV to perform abrupt turns, to help a human operator to better supervise the mission. Additionally, the cost function has a time-dependent component that prevents the MAV from flying above obstacles, reduces progressively its altitude and pushes it towards its starting point, allowing a safe landing within a specified time limit. Figure 4.1b shows an example of a path computed by our algorithm, together with the map acquired during the exploration.

We implemented our approach in C++ using ROS and tested it in simulation as well as in a real indoor environment, along with other state-of-the-art algorithms. We claim that our approach (i) yields a map with a low uncertainty in the probabilistic model, (ii) avoids abrupt changes of direction during the flight, (iii) does not generate a longer path by taking into account the aforementioned aspects, and (iv) is able to compute the next-best-view online and in real-time. Our experimental evaluation backs up these claims and shows that our approach leads to better results compared to some of the current state-of-the-art algorithms.

4.1.1 Information Gain-Based Exploration

Information gain-based exploration is a frequently used approach for exploration. By moving to the unknown space and making observations, the robot acquires a certain amount of new information with its sensors until the whole space is explored and no further significant amount of new information can be obtained. More precisely, information gain-based exploration seeks to select viewpoints resulting in observations that minimize the expected uncertainty of the robot's belief about the state of the world. In this chapter, we focus on autonomous exploration with a MAV with the goal of obtaining an accurate model of the scene. Since the problem of finding the optimal sequence of viewpoints for a complete exploration is NP-complete, it is hard to compute the optimal solution for an exploring MAV online. In order to apply the exploration approach with the available computational resources, we have to make approximations. In this section, we describe the information gain-based approach. In the following sections, we focus on the specific approximations and implementation details of our algorithm.

We describe the uncertainty in the belief of the state of the world through the entropy $H(S)$:

$$H(S) = -\int p(s) \ln p(s) \, ds, \tag{4.1}$$

where S is the state of the world, and s is a possible configuration of the state of the world. Using Equation (4.1), we compute the expected information gain I to estimate the amount of new information obtained by taking a measurement Z while following the path $\mathcal{P}$:

$$I(S, Z^{\mathcal{P}}) = H(S) - H(S \mid Z^{\mathcal{P}}), \tag{4.2}$$

where $\mathcal{P}$ is a collision-free path from the current position to the viewpoint $\mathbf{P}$. Note that $\mathcal{P}$ can be a simple straight line if there are no obstacles along it, or can be computed by a fast, low level path planner such as [82]. The second term of Equation (4.2) is the conditional entropy, defined as:

$$H(S \mid Z^{\mathcal{P}}) = \int p(z \mid \mathcal{P}) \, H(S \mid Z^{\mathcal{P}} = z) \, dz, \tag{4.3}$$

where z is an observation potentially obtained along the path $\mathcal{P}$.

Unfortunately, reasoning about all the potential observations is intractable in nearly all real world applications. Specifically, to obtain the optimal solution it is

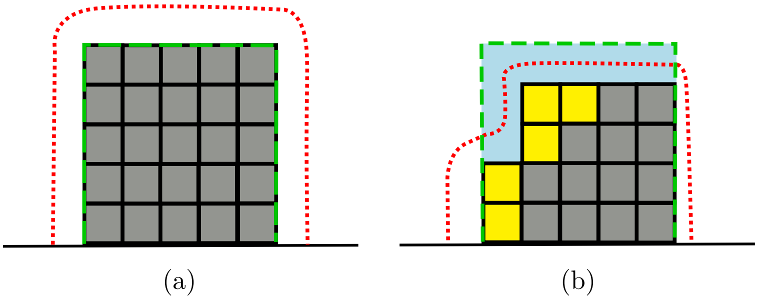


Figure 4.2: (a) The movements of the MAV are constrained to a hull (red dotted line) that is initially built around the user-specified bounding box (blue area delimited by dashed green border). (b) The hull is then adapted according to the new explored (yellow squares) and unknown (gray squares) voxels.

necessary to take into account in advance all the possible sequences of viewpoints, which grows exponentially with the dimension of the measurement space and the number of measurements to be taken. To solve this problem, we approximate the conditional entropy so that it is efficient to compute and we perform the exploration in a greedy fashion by iteratively select only the next best viewpoint.

Additionally, in practice the action of actually recording a measurement has a cost, e.g., the distance that the robot has to travel or the time it takes to reach the new pose from its current one. Therefore, we aim at maximizing the expected information gain I while minimizing the cost of acquiring the new measurement and define to this purpose a utility function U as:

$$U(\mathbf{P}) = I(S, Z^{\mathcal{P}}) - \text{cost}(\mathcal{P}), \tag{4.4}$$

so that selecting the next viewpoint turns into solving:

$$\mathbf{P}^* = \underset{\mathbf{P}}{\operatorname{argmax}}\, U(\mathbf{P}). \tag{4.5}$$

4.1.2 Restricting the Possible Viewpoints

As explained in Section 4.1.1, we require several approximations to achieve online computation. One of the most important issue to address is the selection of a set of next viewpoint candidates. In principle, this set contains the infinite possible poses that a sensor mounted on a MAV can assume in an infinite 3D world.

As a first step, we delimit the space to explore to lie inside a user-specified bounding box, which can for example surround a building or an object. We define a hull that initially surrounds the bounding box (see Figure 4.2a) and we constrain the motion of the MAV to the hull. Whenever the robot has to select a new viewpoint, we dynamically adapt the hull to fit the explored building or object according to the map built so far (see Figure 4.2b). Additionally, we constrain the orientation of the MAV such that the sensor, which has typically a fixed

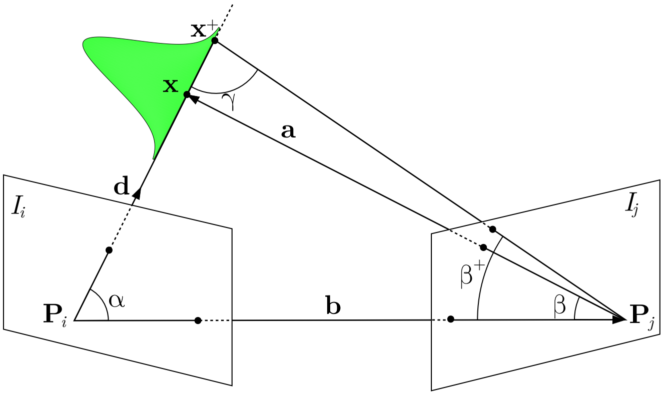


Figure 4.3: The measurement uncertainty of a depth point from two images can be computed through $\mathbf{x}$ and $\mathbf{x}^+$ as elaborated in the REMODE approach [91] for 3D reconstruction from monocular images.

vertical orientation, is always pointing towards the vertical axis passing through the centroid of the bounding box. In this way, we obtain a two-dimensional manifold on which each point is directly mapped to a six-dimensional pose. To compute the set of candidate viewpoints, we sample uniformly a fixed amount of points on the manifold (100 in our experiments, see Section 4.2) every time a new viewpoint needs to be selected. These approximations considerably speed up the computation, but introduce some limitations. In particular, the constrained orientation might lead to a suboptimal exploration in case of stuctures with a particular shape, e.g., an L-shaped building with the concave corner facing towads the centroid of the bounding box and at an excessive distance from it. In this case, alternative solutions can be adopted, but it is important to consider the impact of them on the computational demands.

4.1.3 Measurement Uncertainty

We compute the measurement uncertainty of the sensor as the uncertainty of the depth obtained from two images, using the formulation proposed by Pizzoli *et al.* in their dense reconstruction approach [91].

Given a pair of images I_i and I_j, we compute the variance in the depth estimate of a 3D point $\mathbf{x}$ as

$$\sigma_j^2 = \left(\left\|\mathbf{x}^+\right\| - \|\mathbf{x}\|\right)^2, \tag{4.6}$$

where $\mathbf{x}^+$ is obtained by back-projecting the uncertainty of measuring the point $\mathbf{x}$ in the image plane from image I_j. Figure 4.3 shows an illustration of the estimation.

We then compute the norm of $\mathbf{x}^+$ from the two angles β^+ and γ and the base vector $\mathbf{b}$, see Figure 4.3. To compute these angles, we first need to define the direction $\mathbf{d}$ of the 3D point from I_i, which is defined as

$$\mathbf{d} = \frac{\mathbf{x}}{\|\mathbf{x}\|} \tag{4.7}$$

and can directly be obtained given a calibrated camera. The vector $\mathbf{a}$ from the second projection center to the 3D point is $\mathbf{a} = \mathbf{x} - \mathbf{b}$, exploiting the base vector $\mathbf{b}$ between both camera. This allow us to define the angle α between the base vector $\mathbf{b}$ and the direction $\mathbf{d}$ as

$$\alpha = \arccos\left(\frac{\mathbf{d} \cdot \mathbf{b}}{\|\mathbf{b}\|}\right), \tag{4.8}$$

as well as the angle β between the base vector $\mathbf{b}$ and $\mathbf{a}$:

$$\beta = \arccos\left(-\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}\right). \tag{4.9}$$

Under the assumption that we can localize a point in an image with a standard deviation of σ_{img}, we define the angle β^+, which is defined through β plus the uncertainty of localizing the point in the image mapped into the space of directions:

$$\beta^+ = \beta + 2 \arctan\left(\frac{\sigma_{\text{img}}}{2f}\right), \tag{4.10}$$

where f is the focal length of the camera. Typical feature points can be computed with an uncertainty σ_{img} of 0.3 px to 1 px. In our current implementation, we use a constant uncertainty of 1 px.

As three angles within a triangle need to sum to π, we can compute the third angle γ

$$\gamma = \pi - \alpha - \beta^+, \tag{4.11}$$

and finally $\|\mathbf{x}^+\|$, which in turn is used to compute σ_j^2 according to Equation (4.6):

$$\|\mathbf{x}^+\| = \|\mathbf{b}\| \frac{\sin \beta^+}{\sin \gamma}. \tag{4.12}$$

In practice, we need two views of a 3D point in order to obtain a Gaussian estimate. The uncertainty of the resulting Gaussian, directly depends on the angle γ. The closer γ approaches 90°, the larger the uncertainty reduction.

4.1.4 Approximating the Information Gain

The main goal of our approach is to find the viewpoint that best reduces the uncertainty in the belief about all the points. This uncertainty reduction is given through the expected change in entropy of this belief. Thus, the mutual information for a Gaussian point estimate turns into

$$I(S_j, Z^k) = \frac{1}{2} \ln\left(\frac{\sigma_{x,k}^2 + \sigma_{x,j}^2}{\sigma_{x,k}^2}\right), \tag{4.13}$$

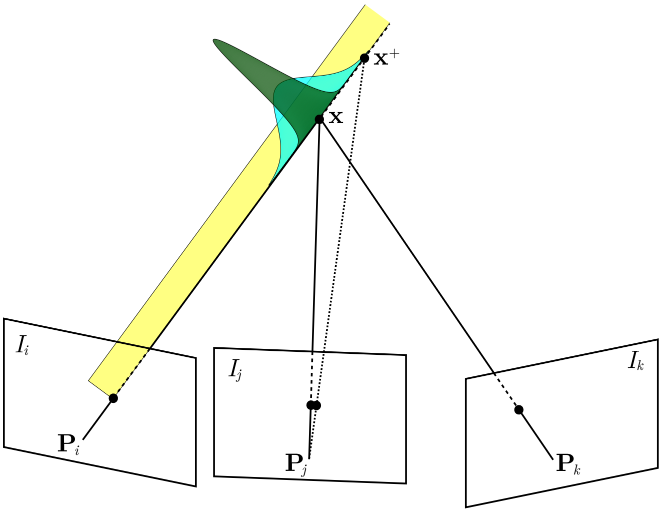


Figure 4.4: Illustration of the reduction of the uncertainty in the estimate about the 3D point location given a third image. The figure depicts the distributions of the uncertanty of the depth of a point estimated using a single image (yellow), two images (cyan) and three images (green).

where Z^k refer to the observations obtained at the camera location $\mathbf{P}_k$, while $\sigma^2_{x,j}$ and $\sigma^2_{x,k}$, computed using Equation (4.6), are respectively the current uncertainty of point $\mathbf{x}$ from view I_j and the estimated uncertainty of the same point from view I_k.

As a result of the Equation (4.13), the expected uncertainty reduction that results from a new image depends on the current uncertainty of the point estimate and on the measurement uncertainty, which itself depends on the geometric configuration of the new viewpoint with respect to the previous viewpoints.

4.1.5 Combining Information from Multiple Measurements

Starting from the formulation of the measurement uncertainty between two images, we can extend it to multiple images. Intuitively, the more views we add, the smaller the uncertainty becomes, i.e., any new observation of the point reduces its uncertainty, which is the main goal of our exploration strategy. Figure 4.4 shows how adding a third image reduces the uncertainty of the measured depth of a 3D point. If the point is seen by a single view of a monocular camera, we can compute the direction of the point, but not its depth. Therefore, the uncertainty is represented by a uniform distribution, shown in yellow in Figure 4.4. After the second view is obtained, the uncertainty of the depth is normally distributed and can be obtained as explained in Section 4.1.3, see the cyan-colored distribution in Figure 4.4. From the third view on, the uncertainty is further reduced, see the dark green Gaussian in Figure 4.4 for an example.

As the correlation between the measurements is unknown, to combine the information from a new image with all the previous ones and compute the total uncertainty, we use the covariance intersection algorithm. Given two distributions of means μ_1, μ_2 and covariance matrices $\mathbf{\Sigma}_1, \mathbf{\Sigma}_2$, it is possible to combine them in a third distribution of mean μ_3 and covariance matrix $\mathbf{\Sigma}_3$, computed as

$$\mathbf{\Sigma}_3 = \left(\omega(\mathbf{\Sigma}_1)^{-1} + (1-\omega)(\mathbf{\Sigma}_2)^{-1}\right)^{-1} \tag{4.14}$$

$$\mu_3 = \mathbf{\Sigma}_3\left((\mathbf{\Sigma}_1)^{-1}\mu_1 + (\mathbf{\Sigma}_2)^{-1}\mu_2\right), \tag{4.15}$$

where ω is a weighting parameter obtained typically by minimizing the determinant or the trace of $\mathbf{\Sigma}_3$. To compute ω, we adopt the closed-form solution proposed by Reinhardt *et al.* [98]. We approximate the distribution resulting from two images to be univariate, assuming the uncertainty only in the depth of the point seen from the first image, which in turn is assumed without uncertainty. Note that, to reduce the computational load, we compute the covariance intersection only between each new view and the first 10 from which the point has been seen.

4.1.6 Storing Information

To store the map of the volume to explore, i.e., the portion of the 3D space contained into the bounding box, we discretize it into voxels. For each voxel, we store the total uncertainty of the portion of space inside of it, as well as the viewpoints from which it has been observed. Note that we treat all the points in one voxel alike, i.e., we compute the expected uncertainty reduction per voxel only. Furthermore, as it is typical for occupancy grids, we assume each voxel independent from the others, therefore the total entropy is the sum of the entropies of each voxel. Voxels that are still unexplored have the maximum uncertainty, as no information exists about the state of the cell.

Each time the algorithm needs to choose a new viewpoint, it computes the total expected information gain of each candidate viewpoint by casting rays from that point and summing the expected information gain of all the voxels hit by the rays. Note that, since we acquire images at a constant framerate, for each candidate viewpoint we evaluate the information gain for all the intermediate images, by subsampling a computed path between the current position and the possible next one. When a new observation is obtained, we update the uncertainty stored in each voxel seen by the new image. We efficiently store the voxelized map using octrees with the C++ library Octomap [52]. This library can represent voxels with three states (occupied, free, and unknown) in an occupancy grid. The API of Octomap allows for automatically updating the states of the voxels by using ray casting, given a depth image and the sensor's pose. In our implementation we use a maximum resolution of $0.125\,\mathrm{m}^3$, as we discretize the space into voxels with a size of $0.5\,\mathrm{m} \times 0.5\,\mathrm{m} \times 0.5\,\mathrm{m}$.

4.1.7 Changes in the Direction of Flight

As mentioned in Section 4.1.1, the action of actually recording the next measurement has a cost, represented by the second term of Equation (4.4). We define this cost by taking into account the particular needs of an exploration algorithm for MAVs, as explained in this section and in the following one. A relevant problem of using autonomous MAVs is that in most countries a human operator is required to supervise the mission at all times, to intervene in case of malfunction or other emergencies. To this purpose, it is helpful if the operator is able to predict the trajectory of the robot, so that he or she can promptly take control of the MAV if it is about to perform a dangerous action. According to our experience, this is substantially simpler if the MAV avoids erratic motion, i.e., its trajectory is short and avoids abrupt changes of direction. Our approach achieves such a behavior by using the cost function:

$$\mathrm{cost}(\mathcal{P}) = d(\mathcal{P}) + \theta(\mathcal{P}), \tag{4.16}$$

where $d(\mathcal{P})$ grows linearly with the length of the path $\mathcal{P}$ between the current location of the robot and point $\mathbf{P}$, while $\theta(\mathcal{P})$ is a function that grows linearly with the maximum change in orientation that must be executed by the MAV along the path.

4.1.8 Time-Dependent Cost Function

Most available MAVs have a limited battery life. When planning a mission, it is essential to take the expected battery life into account to prevent the drone from crashing and eventually to allow a safe landing at the starting point. We achieve this by adding a time-dependent component to our cost function. Given a user-specified time limit for the mission, the algorithm dynamically computes a critical time value t_{critical}, based on the trajectory needed to move the MAV from the current location to the starting point. In our implementation, this trajectory is computed by a low-level planner [82]. Once the elapsed-time reaches t_{critical}, the algorithm adds the starting point to the list of possible next points and the time-dependent cost function activates. This has the effect of favoring the robot to:

- fly towards the starting point (for landing),
- avoid flying above obstacles to allow for a potential emergency landing, and

- fly closer to the ground to prevent possible impacts if the battery dies.

Note that, since the starting point is now part of the set of candidate next points, the MAV will land on it as soon as it is close enough.

Thus, we extend Equation (4.16) by adding the time dependent components, and we obtain the new cost function:

$$\mathrm{cost}(\mathcal{P}, t) = d(\mathcal{P}) + \theta(\mathcal{P}) + T(t)\big[f(t) + d_{\text{start}}(t) + z(t)\big], \tag{4.17}$$

where:

- $T(t) = \begin{cases} 1 & \text{if } t \geq t_{\text{critical}} \\ 0 & \text{if } t < t_{\text{critical}} \end{cases}$

- $f(t)$ grows linearly with the elapsed time if in the map there are occupied voxels (excluding the ground) in the area below the MAV, it is equal to 0 otherwise;

- $d_{\text{start}}(t)$ grows linearly with the elapsed time and is proportional to the distance between the current position and the starting point;

- $z(t)$ grows linearly with the elapsed time and is proportional to the current altitude of the MAV.

These terms can be chosen in different ways. In our implementation, we represent the cost function as a weighted sum of the different terms, where the weights are tuned by hand. The functions $f(t)$, $d_{\text{start}}(t)$, and $z(t)$ have dynamic weights that range between w_{min} and w_{max} and are computed as:

$$w(t) = \left(\frac{t_{\text{elapsed}}}{t_{\text{max}}} - 0.8\right)\frac{w_{\text{max}} - w_{\text{min}}}{0.2} + w_{\text{min}}. \tag{4.18}$$

Table 4.1 shows the weigths that we use in our experiments. Note that, since the robot might still explore unknown space, this approach does not *guarantee* its return to the starting point, but the whole process allows the MAV to increase its safety while still maximizing the information gain. The cost function in Equation (4.17), plugged in Equation (4.4), contributes together with the information-gain to the utility fuction we use to select the next best view.

4.2 Experimental Evaluation

The main focus of our exploration algorithm is to select viewpoints for the purpose of an accurate 3D reconstruction of the environment. Therefore, each new viewpoint has the purpose of exploring new voxels and reducing the uncertainty of

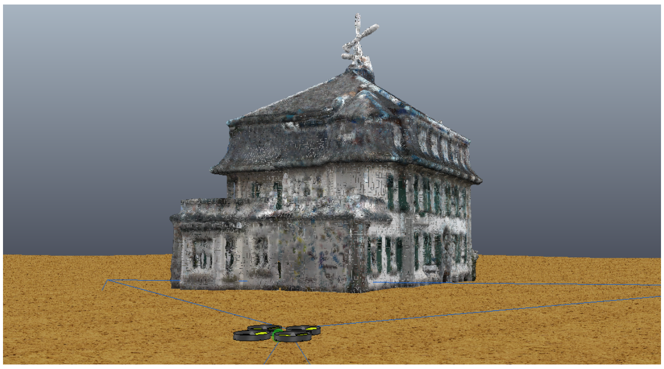


Figure 4.5: Simulated environment in V-REP simulator. The scene contains the Frankenforst building and the MAV.

Table 4.1: Weights used in the cost function in our implementation.

Function	Weight [s]
$d(\mathcal{P})$	1000
$\theta(\mathcal{P})$	30000
$f(t)$	$w_{\text{min}} = 5000$, $w_{\text{max}} = 6000$
$d_{\text{start}}(t)$	$w_{\text{min}} = 5000$, $w_{\text{max}} = 12000$
$z(t)$	$w_{\text{min}} = 300$, $w_{\text{max}} = 4000$

the already observed ones. Additionally, the selected viewpoints favor the MAV to follow a path free of sharp changes of direction to allow a human operator to predict more easily its trajectory. Furthermore, the algorithm takes the time into account for a safer flight and landing when the robot is about to run out of battery. Finally, our system has to be fast enough for a real-time execution, as the next best view is selected online while the MAV is flying.

Our experiments are designed to show the capabilities of our method and specifically to support the key claims we made in the introduction, which are: (i) our approach yelds a map with a low uncertainty in the probabilistic model, (ii) it avoids abrupt changes of direction during the flight, (iii) it does not generate a longer path by taking into account the aforementioned aspects, and (iv) it is able to compute the next best viewpoint in real-time and online on an exploring system.

We furthermore provide comparisons to two recent state-of-the-art methods: the exploration algorithm based on Proximity Count VI proposed by Isler *et al.* [55] and the one proposed by Vasquez-Gomez *et al.* [135]. We used the existing open source implementation by Isler *et al.* [55] of both the algorithms, while we implemented our approach from scratch using C++ and ROS.

4.2.1 Experimental Setup

We tested the three approaches with identical settings on a simulated environment, using the V-REP simulator by Coppelia Robotics. We set all the parameters for the algorithms to the default values provided by Isler *et al.* [55], with the exception of the camera calibration, the ray caster resolution (reduced by a factor of 2), and the ray caster range of 20 m.

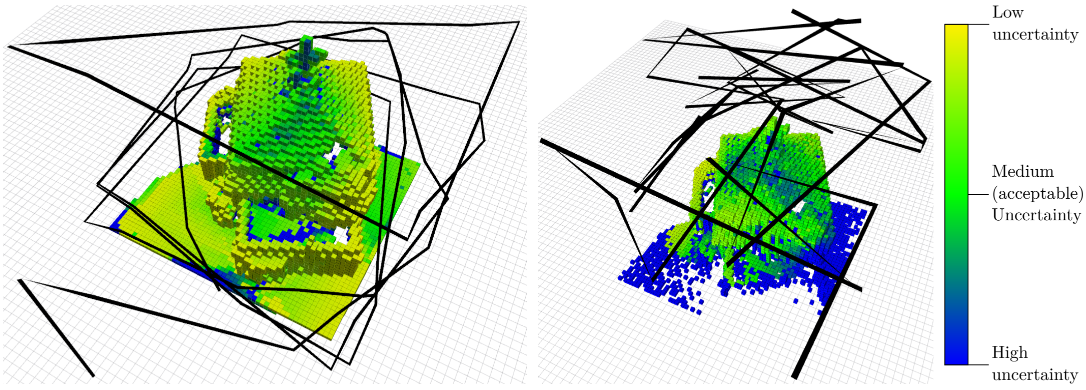
The robot in our simulation is a quadcopter with a camera mounted on the front and facing downwards with an angle of 45°, as typically vision sensors on MAVs are mounted with angles ranging from 0° to 90°. The camera has a field of view of 86°, a resolution of 2040 by 2040 pixels and a constant framerate of 20 Hz. We assume to know the pose of the sensor, that in practice can be obtained from a SLAM system such as the one proposed by Schneider *et al.* [108], which runs at 100 Hz on a MAV and operates with an uncertainty of few centimeters. As our algorithm is independent of the sensor used, we obtain the depth information directly from the simulator instead of computing it from the images acquired by the camera. In a real case scenario, this information can come for example from a structure from motion algorithm, or from a stereo camera.

As the object to explore, we selected a building at the University of Bonn called Frankenforst (Figure 4.1a), and we imported in the virtual scene a 3D model of such building obtained from terrestrial laser scans, see Figure 4.5. We specified a bounding box around the bulding with a size of 29 m × 32 m × 25 m, which delimits the volume to explore. As starting locations for the MAV, we sampled 10 locations equally spaced on a circle on the ground around the building. As our algorithm has a random component, we performed every test 10 times, with the robot starting from each of these 10 locations, performing a vertical takeoff and then starting the exploration mission.

During the execution of our algorithm, every time a new viewpoint had to be selected, we recomputed the hull representing the action space of the robot at a distance of 5 m from the occupied voxels in the map and we sampled 100 points on it (see Section 4.1.2). Note that for the comparison with the existing methods [55, 135], we disabled our time-dependent cost function and introduced a boolean cost function in the other two algorithms to prevent collisions, as the implementation by Isler *et al.* [55] does not provide a functioning collision avoidance system. We stopped the three algorithms after 40 viewpoints were approached.

4.2.2 Precision of the Reconstruction

The first evaluation is designed to support the claim that our approach selects viewpoints that increase the number of observed voxels and reduce the uncertainty



(a) Our Approach (b) Vasquez-Gomez *et al.*

Figure 4.6: Result of the execution of our algorithm (a) and the approach by Vasquez-Gomez *et al.* [135] (b) after 40 computed viewpoints. The black line represents the trajectory and the model in the center is the result of the mapping. The voxels are colored according to their uncertainty.

of the already observed ones. We evaluate the precision of the 3D reconstruction by measuring the uncertainty of the observations obtained from the camera, using the formulation explained in Section 4.1.3.

Figure 4.6a shows the path and the map resulting from the execution of our algorithm after 40 computed viewpoints. Each voxel is colored according to the total uncertainty of the points it contains, with the colormap shown in Figure 4.6. For an accurate 3D reconstruction, the uncertainty should be "medium" or lower (from green to yellow in our colormap). The figure shows qualitatively that most of the voxels in the map have an uncertainty lower than the acceptable value, i.e., it is possible to accurately reconstruct the building in 3D from the acquired images.

As a quantitative evaluation, Figure 4.7 and Figure 4.8 show how the normalized global uncertainty and the total number of explored voxels in the map evolve during the exploration. The figure shows mean values and standard deviations over our 10 tests described above. We define the normalized global uncertainty as:

$$\sigma^2_{\text{global}} = \frac{\sum_{i=1}^{N} \sigma_i^2}{\sigma^2_{\text{max}}}, \tag{4.19}$$

where σ_i^2 is the uncertainty of the i-th voxel, N is the number of voxels in the map and σ^2_{max} is the maximum possible uncertainty, i.e., the sum of the uncertainty of every voxel before the exploration starts. At the beginning of the exploration, every voxel has infinite uncertainty, simulated in practice by a large fixed value (in our implementation 10,000,000). Note that the metric we use depends on this maximum value. Therefore, we set the value identically for all algorithms so that the obtained performance measure allows for a fair comparison between

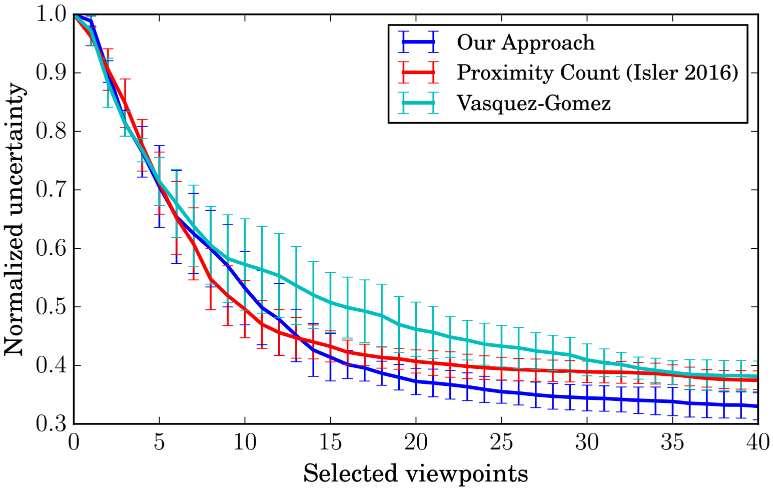


Figure 4.7: Global uncertainty of the map at each selected viewpoint (mean values and std. deviations).

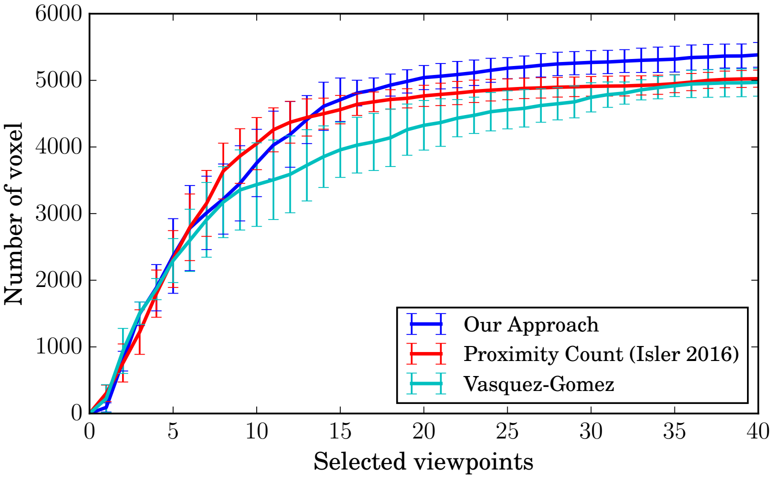


Figure 4.8: Total number of explored voxels in the map at each selected viewpoint (mean values and std. deviations).

the approaches. Moreover, since we use an octree to store the map, to compute this measure we expand the tree to obtain all the voxels on the same level, i.e., we convert the map to a simple voxel grid for this evaluation. Both Figure 4.7 and Figure 4.8 show an advantage of our approach. Moreover, by looking at the std. deviations it is clear that none of the algorithms are affected by the starting point of the flight, and in general the results are consistent among different flights.

4.2.3 Path Smoothness

A particular focus of our approach is the shape of the flight path. From our experience, a human supervisor, who is required by law in several countries, can more easily predict the trajectory of the MAV if it is free of abrupt turns. Our second evaluation supports our second claim, namely that our approach is

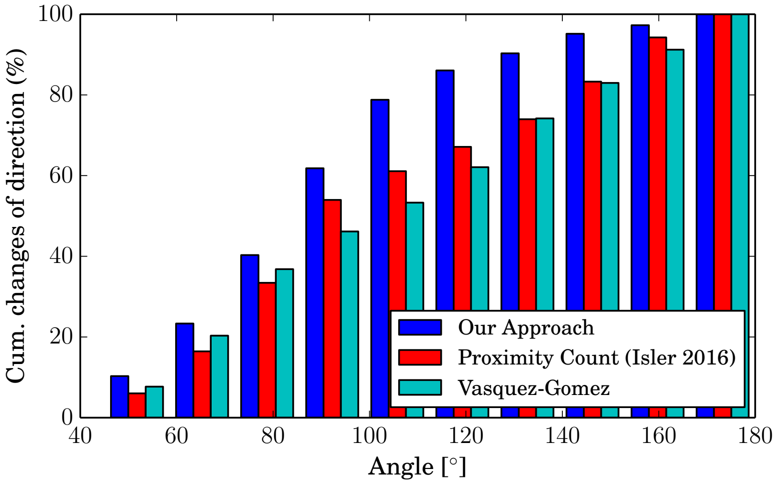


Figure 4.9: Cumulative histogram of the changes of direction.

able to select viewpoints that favor a path that avoids sharp changes of direction. This is qualitatively visible in Figure 4.6, where the path chosen by our approach, compared to the one by Vasquez-Gomez *et al.* [135], is fairly regular and basically performs larger turns only if the model requires it.

To compare this aspect between the different algorithms, we considered the angles of each change of direction, and counted them. The histogram in Figure 4.9 shows the cumulative percentage of changes of direction for angles between 40° and 180°, divided in 10 intervals. As the figure shows, the other approaches have no preference at all when choosing the direction to take for the next viewpoint (the bars have a linear trend). Our algorithm, on the other hand, chooses in 80% of the cases a point at an angle below 100°, thanks to our utility function, which takes this measure into account.

4.2.4 Path Length

An important aspect of an exploration algorithm is the length of the path that the robot needs to follow to complete the procedure. In this section, we support our third claim, i.e., our algorithm has performances aligned to the current state of the art in terms of path length, despite its utility function, which is designed for a more predictable path.

Figure 4.10 shows the total path length after approaching each next viewpoint. The best performing algorithm in this case is the one by Vasquez-Gomez *et al.*, due to its utility function, specifically designed to create overlaps between the views. Our approach, instead, is aligned to the one by Isler *et al.* We also consider the uncertainty reduction and relate it to the travelled distance, see Figure 4.11. The plot shows that after 350 m the three algorithms show a similar performance, with a slight advantage of our approach. Therefore, the longer distance between

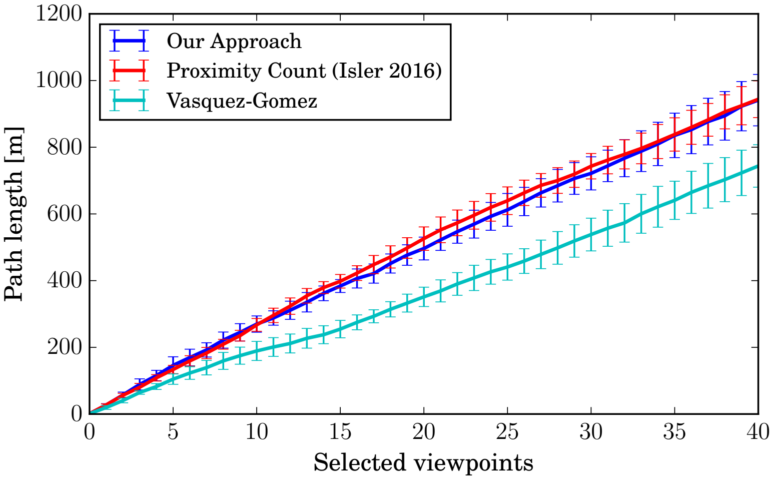


Figure 4.10: Path length at each selected viewpoint.

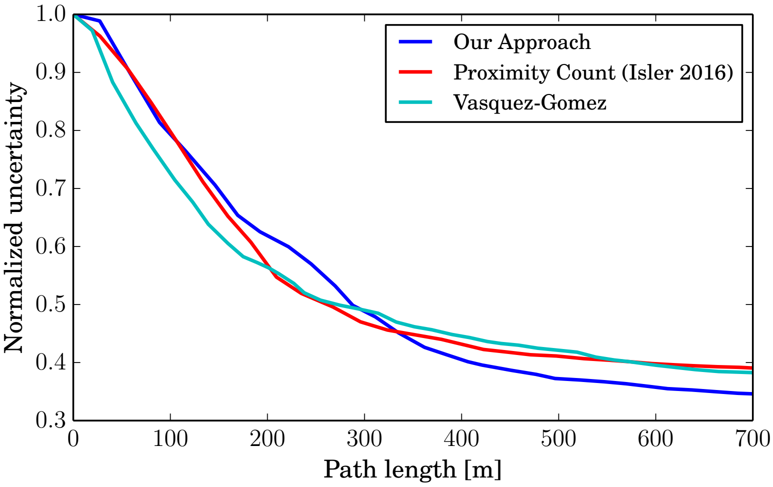


Figure 4.11: Map uncertainty versus traveled distance.

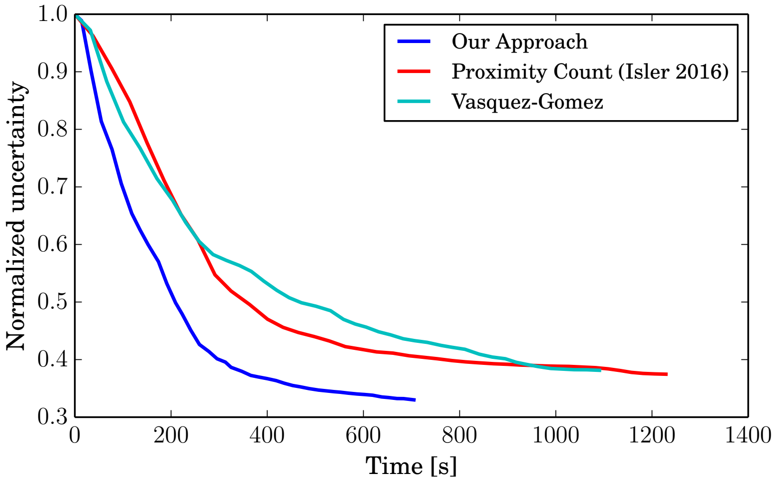


Figure 4.12: Map uncertainty versus elapsed time.

Table 4.2: Average time and std. deviation to compute a viewpoint with the different algorithms.

Algorithm	Average time and std. deviation [s]
Our algorithm	1.31 (±0.41)
Proximity count [55]	6.53 (±1.56)
Vasquez-Gomez	6.58 (±1.56)

the single viewpoints is compensated by a better reduction of the uncertainty. Thus, we can conclude that our strategy to select viewpoints for a more regular path does not substantially affect the total length.

4.2.5 Execution Time

The final comparison we make is between the execution time of the three approaches. Table 4.2 shows the average time spent to compute the next viewpoint on a single core of a regular Intel Core i7 CPU. In our tests, our approach performed about 5 times faster. In addition to that, Figure 4.12 shows the uncertainty against the elapsed time. As can be seen, our approach reduces the uncertainty faster while the time elapses and it is fast enough to be used in real-time.

4.2.6 Time-Dependent Cost Function

The last aspect of our algorithm is its capability of selecting viewpoints that favor the MAV to move back to its starting point before it runs out of battery, while still taking into account the information gain expected from the new viewpoints. Figure 4.13 shows a path computed with a time limit. At first, the algorithm explores normally the building (black continuous path), but when the critical

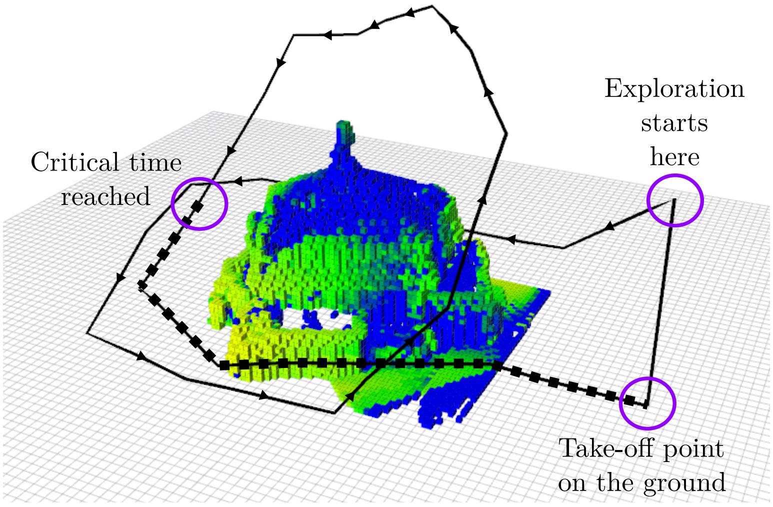


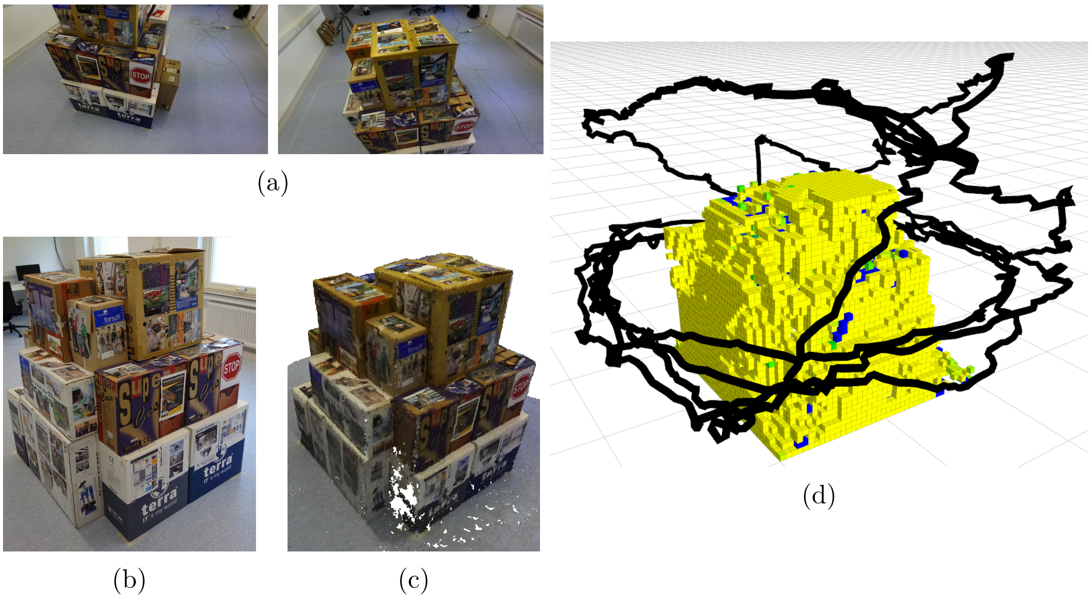
Figure 4.13: Path computed with the time-dependent cost function enabled. The square markers represent the path generated after the critical time, which moves the MAV back to the starting point.

time is reached, as shown by the square markers, the MAV flies towards the starting point, while still trying to reduce the uncertainty, keeping a low altitude and avoiding flying above obstacles.

4.2.7 Real World Experiment

We finally provide a test of our algorithm in the real world. We created a test environment composed by a structure of boxes in an indoor scene, see Figure 4.14b. We use a ZED stereo camera, which provides, out-of-the-box and in real time, the depth information computed from the stereo images. We run our algorithm with similar settings as the simulated tests, eventually scaled to the size of the new environment. In particular, we used the same weights for the cost function (see Table 4.1) and sampled the same number of candidate next viewpoints at each iteration (100), but we increased the map resolution by reducing the voxel size to 0.05 m per side, and we reduced the distance of the hull (representing the action space of the robot) from the occupied voxels in the map to 1 m. Figure 4.14a shows some frames acquired by the camera during the execution of the algorithm. Figure 4.14d shows the final map of the structure and the actual trajectory of the camera, after 60 computed viewpoints.

To test the use of our algorithm for a real 3D reconstruction case, we used the recorded image sequence as input for an offline, out-of-the-box dense reconstruction approach. Figure 4.14c shows the final dense pointcloud. Note that no contribution for the dense reconstruction approach itself is claimed here.



(a)

(b)

(c)

(d)

Figure 4.14: Real world experiment: (a) Two examples of camera frames. (b) Test environment. (c) Obtained model. (d) Voxel map and followed path.

4.3 Related Work

The aim of the information-gain exploration problem is to select actions for the robot that lead to informative measure from its sensors. A popular approach to this problem is frontier-based exploration [142], which consists of selecting the next viewpoint on the boundary between the known free space and the unexplored space. This technique is widely used for 2D exploration, for example by Bourgault *et al.* [14], who focus on maximizing localization accuracy, Stachniss *et al.* [118], who represent the posterior about maps and poses using Rao-Blackwellized particle filters or, more recently, Perea Ström *et al.* [124], who additionally make predictions about the structure of the environment using previously acquired information. Frontier-based exploration is less popular when it comes to 3D exploration with MAVs, but some recent works have shown promising results. Fraundorfer *et al.* [43], for example, combine a frontier-based approach with a Vector Field Histogram+ algorithm and the Bug algorithm to efficiently explore both in dense and sparse environments. Shen *et al.* [112] identify frontiers using a stochastic differential equation that simulates the expansion of a system of particles with Newtonian dynamics to explore indoor environments. Cieslewski *et al.* [23], instead, extend the classical frontier-based approach for rapid exploration. The idea is to generate velocity commands to reach frontiers in the current field of view, instead of planning trajectories as it is typical for this approach. A related, but different approach is the one proposed by Bircher *et al.* [13], who employ a planner based on a receding horizon, using a random tree.

When applied to vision sensors for 3D reconstruction, the problem of exploration is commonly called active vision [3], active perception [9], or next best view [110]. Hoppe *et al.* [48] particularly focus on recostruction and propose a stucture-from-motion algorithm that provides, in real time, a visual feedback to a human operator. Dunn *et al.* [33], instead, start from an existing model and aim at refining it by choosing the next best view that best reduces its uncertainty. They represent the structure with adaptive planar patches and combine the covariance matrices of the patches and the texture properties of the object to select the next best view. Kriegel *et al.* [67] require less previous knowledge and focus on the completion of 3D objects. They employ a laser scanner and their algorithm is based on surfaces and not on volumetric representations. They select the next best view by taking into account the boundaries of the surface and the occluded areas. Pito [90] also presents a surface-based algorithm to map 3D objects with a range scanner. The general idea is to select the next best view that covers as much unseen space as possible while satisfying certain overlap constraints. Quin *et al.* [96] propose an alternative application of a next-best-view algorithm, targeting the full exploration of an environment with an RGB-D sensor mounted on a manipulator in a fixed position in the environment itself. The algorithm favors points that are close to the current one in the configuration space of the robot as possible next best viewpoints. Kranin *et al.* [66], on the other hand, consider a fixed sensor and use a manipulator to move an object in front of it. Their next-best-view algorithm is information-gain based, but additionally takes into account the manipulator occlusions. Potthast and Sukhatme [93] target cluttered environments, and propose a solution based on probabilistic methods to compute the potential information gain of the candidate viewpoints. Trummer *et al.* [133], in contrast to most of the previous approaches use an intensity camera instead of a range scanner. They mount the camera on a manipulator and compute the optimal next best view on a sphere around the object of interest.

Most of these approaches focus on recostruction of small objects, while we are interested in large scenarios such as the recostruction of a building. One possibility is to perform the computations offline given some a priori information of the environment. Bircher *et al.* [12] propose an approach for computing a path to explore a large building with an MAV. They solve a traveling salesman problem to find the best path that connects the viewpoints that are sampled by knowing the mesh of the building to explore. Their focus is on inspection path planning, meaning that they are not interested in exploring an unknown environment, but rather in covering a known one. Schimd *et al.* [107], similarly, exploit a previously known 2.5D model of the environment to compute every non-redundant viewpoint and solve a traveling salesman problem to minimize the path length. However, they assume the given model to be coarse and their goal is, similarly to ours, an accurate 3D reconstruction from the images acquired during the exploration.

Another interesting approach is the one proposed by Schade *et al.* [111], who explore the environment by planning a path, using the gradient of a harmonic function based on the boundaries between known and unknown space on a 2D plane of a 3D occupancy grid.

The vast majority of the state of the art consists of small variations of the main techniques described above. For example, Haner *et al.* [49] do not only compute the next best view, but also take into account a whole set of future imaging locations. Mostegel *et al.* [77] focus more on localization stability rather than reconstruction and Sadat *et al.* [103] focus on maximizing feature richness. Similarly to our approach, many greedy information gain-based exploration approaches boil down to an algorithm that samples candidate viewpoints, computes a utility function that takes into account the expected information gain and other factors, and selects the viewpoint with the maximum utility function. Delmerico *et al.* [29] describe a general framework for this kind of methods and compare different approximations for the information gain, such as the one by Vasquez-Gomez *et al.* [135], which optimizes the overlap between views and the angle of the sensor w.r.t. the surface, while also taking into account occluded areas. Forster *et al.* [41] use a similar approach, but additionally consider the texture of the explored surface in their approximation of the information gain. Bai *et al.* [8] also employ a similar method, but select the next best view using a deep neural network, to avoid the commonly used (but computationally expensive) ray casting for computing the approximated information gain. In contrast to that, Charrow *et al.* [18] use a two step approach, consisting in generating a set of trajectories, choosing the one that maximizes the information-theoretic objective and refining it with a gradient-based optimization.

Finally, there are several approaches that do not focus on 3D reconstruction, but on different goals, such as Freundlich *et al.* [44], who plan the next best view to reduce the localization uncertainty of a group of stationary targets, Atanasov *et al.* [7], who focus on decentralized multi-sensor exploration or Strasdat *et al.* [123] who address the problem of which landmark is useful for effective mapping.

4.4 Conclusion

In this chapter we presented a novel approach for vision-based autonomous exploration on MAVs. Our approach iteratively samples candidate viewpoints and greedily selects the optimal one using a utility function. This function takes into account the expected information gain from the new view, the distance from the current position to the new point and the change of direction of motion. Additionally, a time-dependent component in the function allows for a safer flight and landing when the MAV is about to run out of battery. We implemented our approach in C++ and ROS and evaluated it in simulated and real environments. We compared the approach to two state-of-the-art methods and our experiments show that our algorithm allows for a precise exploration while flying on a trajectory comparably free of abrupt changes of direction of motion, without sacrificing the path length and in an online, real-time fashion.

Part II

Non-Static Environments

Chapter 5

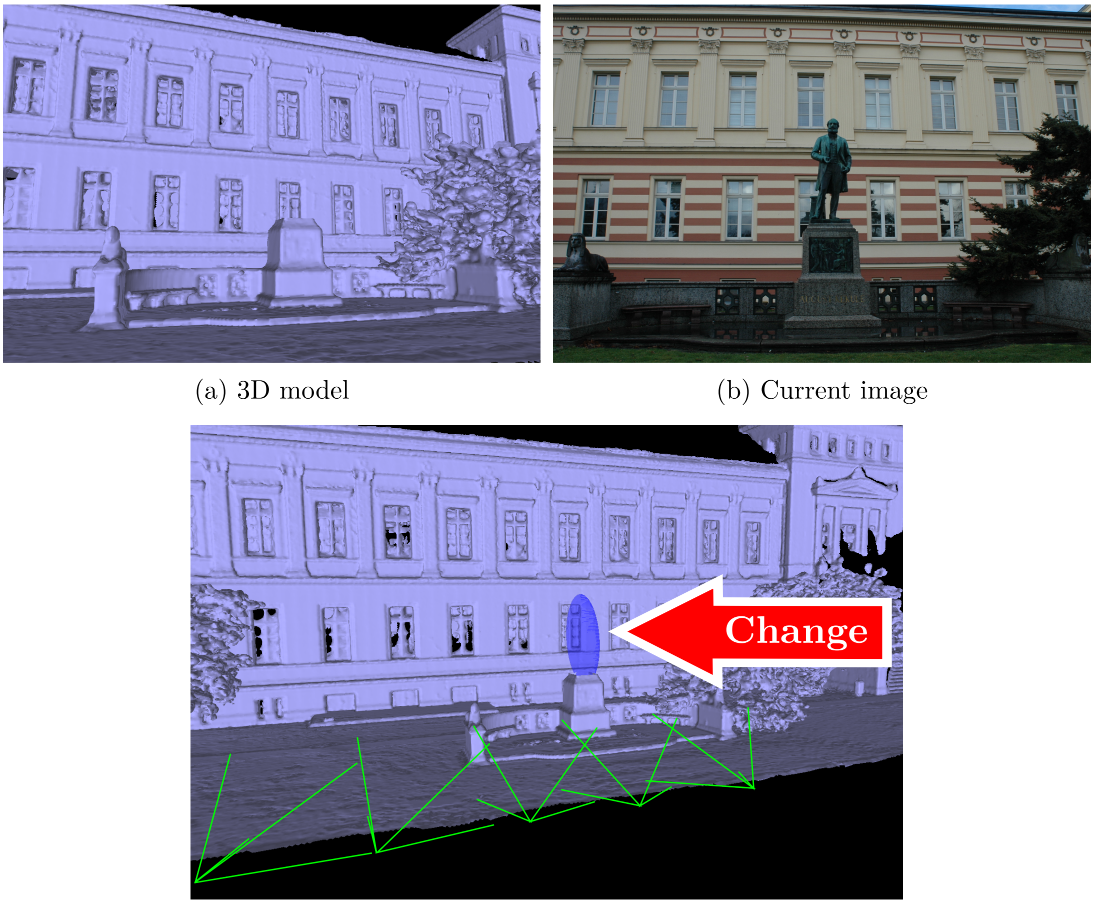
Change Detection in Non-Static Environments

Building 3D models of the environment is a frequently addressed problem in robotics as they are needed for a wide range of applications. For most applications that include autonomous behavior, such models should correspond as well as possible to the current state of the environment. In the first part of this thesis, we focused on how to build such models from the acquired sensor data and how to select the best viewpoints for acquiring such data autonomously. However, we always assumed a static environment, while in practice this is not the case. In case the environment changed substantially, existing models must be updated. For this purpose, the possibility of directing a mapping or exploring robot directly towards the possible regions that have changed instead of repeating the whole mapping process is advantageous. Therefore, it is important to reliably identify locations in a 3D model that have changed.

In this chapter, we address the problem of finding changes between a previously built 3D model and its current state based on a short sequence of keyframe images recorded in the environment, see Figure 5.1 for an illustration. Two aspects are important here: first, we want to reliably locate changes in the model and second, the approach should have a limited computational demand so that it can be executed on a mobile platform, allowing an exploring or mapping robot to plan its next action according to the location of change.

5.1 Image-Based Geometric Change Detection

The main contribution of this chapter is a new and fast approach for identifying differences between an existing 3D model and a short sequence of images recorded in the environment. For finding inconsistencies, we do not build a new 3D model



(a) 3D model

(b) Current image

(c) Result

Figure 5.1: Our approach aims at quickly finding changes in the environment based on an existing 3D model and a sequence of (currently recorded) images.

from the newly obtained image data and compare the result to the existing one. Instead, we back-project the currently obtained image onto the 3D model and then project it to a viewpoint at which another image of the current sequence has been taken. Through a comparison between the re-projected images and the one observed in reality, we can identify possible regions of change. To eliminate ambiguities, this process is executed for multiple image pairs. Typically, 4-5 keyframe images are sufficient to find areas of change and then estimate the 3D location where the geometry has changed. Compared to existing approaches for visual change detection such as the work by Taneja *et al.* [130] or Ulusoy *et al.* [134], our method is substantially faster towards execution on a mobile robot. Note that our approach only compares images from the current sequence with each other, i.e., no current image needs to be compared to an old one. This makes our approach robust to seasonal changes, weather conditions, or any changes that are usually present when taking image sequences at two distinct points in time.

Our approach identifies the approximate area of change fast enough to be executed on a navigating robot, which sets it apart from several related other techniques. We identify inconsistencies by comparing the acquired images to re-projected images that would have been obtained assuming the 3D model is correct, in combination with a forward intersection of the potentially inconsistent regions. We implemented our approach in C++ and tested it on existing datasets and on our own. Our experiments show that our method quickly finds the approximate location of the change in the scene and is fast enough to potentially guide an exploring ground robot or UAV seeking to map the changes in the environment. We publicly share both our open source implementation[1] and our own dataset including the 3D models[2].

We make two key claims: our approach is able to (i) identify the location of changes in the environment, in the form of 3D volumes in the world coordinate frame, using a 3D model and a sequence of images, and (ii) it is fast enough to be executed on a mobile robot, i.e., analyzing a sequence of keyframe images does not take longer than recording it, e.g., a few seconds for a sequence of five keyframe images.

Our approach aims at spotting areas in an environment that have changes with respect to a previously built 3D model. It does so by exploiting a sequence of around five images through evaluating how the projections of image content from one image to the model and back to another image look like. In terms of computational demands, this process is substantially more efficient than generating a new, dense 3D model and comparing it directly with the given one. The first step is to detect possible inconsistencies of an image with its neighboring images assuming that the 3D model is correct. After computing pairwise inconsistency hypotheses, we fuse them to eliminate the intrinsic ambiguities and estimate the location of change by triangulation. Given that we look for inconsistencies between the 3D model and new images, our approach only finds changes from images where the rays corresponding to pixels intersect with the 3D model.

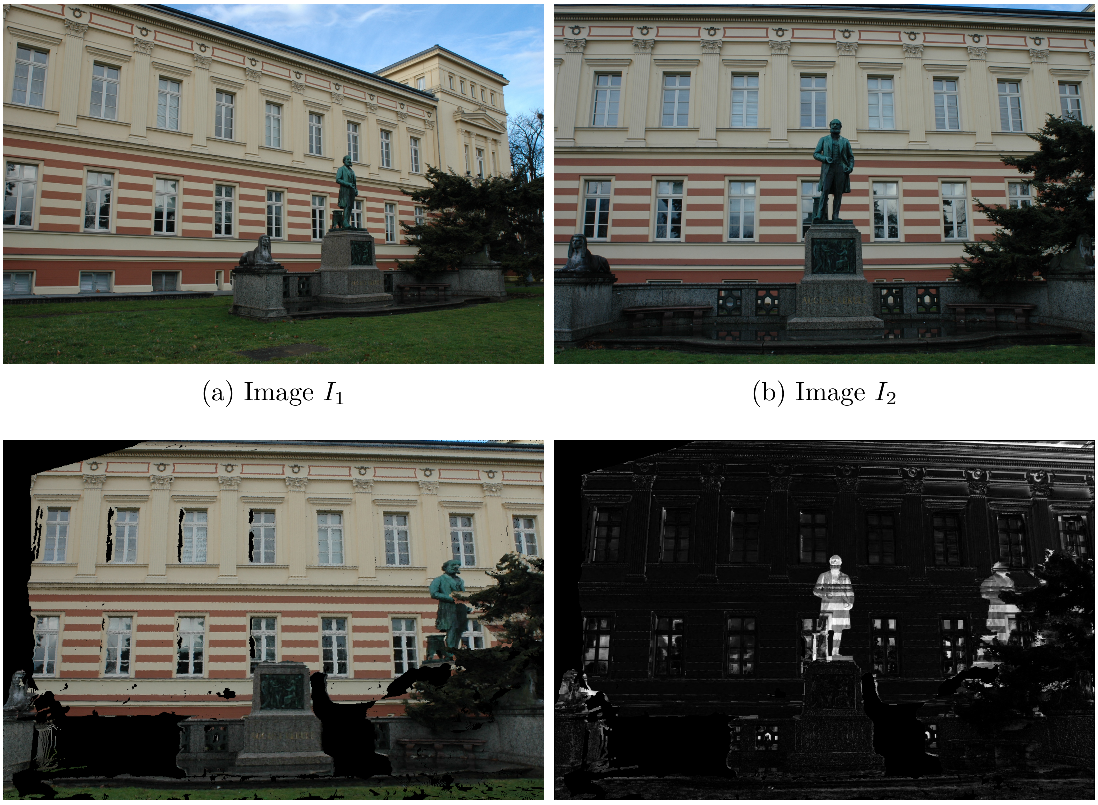
Note that in this chapter we assume a good pose estimate for the camera. We obtain the (approximate) location of the 3D model and the viewpoint of the images as described in Section 5.1.1 below.

5.1.1 Camera Pose Estimate

Our algorithm requires an estimate of the viewpoints of the images w.r.t. the 3D model. We obtain this through direct georeferencing fusing GPS, IMU, and visual odometry, as described in [108]. The approach employs the iSAM2 algorithm, and provides uncertainty information about all sensor poses in the form of a covariance

[1] https://github.com/PRBonn/fast_change_detection

[2] http://www.ipb.uni-bonn.de/data/changedetection2017



(a) Image I_1

(b) Image I_2

(c) Re-projection of I_1 onto I_2

(d) Inconsistencies

Figure 5.2: A pair of images, the first image re-projected onto the second, and the inconsistencies between them.

matrix. In case no GPS information is available, approaches for camera to 3D model localization such as [17] can be used, although we did not directly try that here.

5.1.2 Inconsistencies Between Image Pairs

Through this chapter, as it is critical to distinguish between 2D and 3D entities, we use a slightly different notation compared to the rest of this thesis. In particular, we write with capital letters not only matrices, but also some of the vectors representing 3D entities.

Given the calibration matrix and the pose at which the camera took an image I, we can compute the projection of an arbitrary 3D point $\mathbf{X}_{\text{world}}$ onto the image plane resulting in a 2D point at pixel $\mathbf{x}$:

$$\mathbf{x} \quad = \quad \mathbf{P}\mathbf{X}_{\text{world}}, \tag{5.1}$$

where $\mathbf{x}$ is expressed in homogeneous coordinates and $\mathbf{P} = [\mathbf{KR}|-\mathbf{KRt}]$ is the 3×4 camera projection matrix composed by the 3×3 matrix $\mathbf{KR}$ and the 3×1

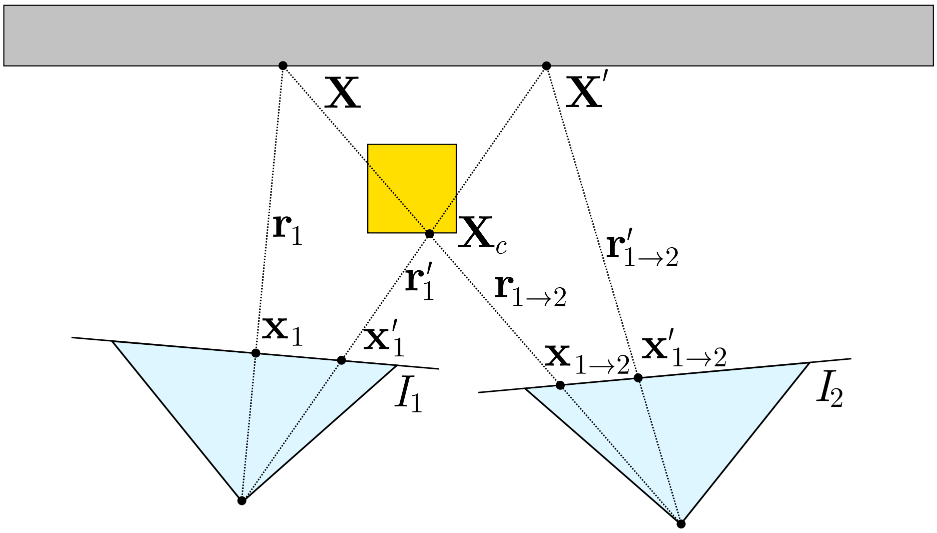


Figure 5.3: Re-projection procedure. The gray rectangle represents the known 3D model, while the yellow square is a change not present in the original model. Using two images, a point $\mathbf{X}_c$, not present in the model, is re-projected onto two pixels $\mathbf{x}_{1\to 2}$ and $\mathbf{x}'_{1\to 2}$.

vector $-\mathbf{KRt}$, with $\mathbf{K}$ the calibration matrix of the camera, $\mathbf{R}$ and $\mathbf{t}$ the rotation and translation that transform the world coordinates into camera coordinates.

By inverting Equation (5.1), we compute the ray from the projection center of the camera through the pixel to the 3D world. This allows us to back-project each pixel of I onto the 3D model assuming the known intrinsic parameters $\mathbf{K}$ and the rotation matrix $\mathbf{R}$ from the extrinsic parameters:

$$\mathbf{r} \;=\; \mathbf{R}^{\mathsf{T}}\mathbf{K}^{-1}\mathbf{x}, \tag{5.2}$$

where $\mathbf{r}$ is the direction of the ray in world coordinates.

To detect inconsistencies between a pair of images consisting of the images I_1 and I_2, we create a new image $I_{1\to 2}$ that represents the content of I_1 as seen from the view point of I_2 given the 3D model. Given that we know, from Equation (5.2), the view direction $\mathbf{r}_1$, we compute the intersections $\mathbf{X}$ between the rays and the 3D model and project $\mathbf{X}$ onto the image plane of I_2 to obtain $I_{1\to 2}$ (see Figure 5.2c for a real example):

$$\mathbf{x}_{1\to 2} \;=\; \mathbf{P}_2\,\mathbf{X}, \tag{5.3}$$

where $\mathbf{P}_2$ is the camera projection matrix corresponding to image I_2. In this way, we obtain a new image $I_{1\to 2}$ that can be compared with I_2. Since the *exact* poses of the cameras are unknown and the 3D model is not perfect, the point $\mathbf{x}_{1\to 2}$ has an uncertainty represented by the covariance matrix $\mathbf{\Sigma} := \mathbf{\Sigma}_{\mathbf{x}_{1\to 2}\mathbf{x}_{1\to 2}}$. To consider this uncertainty, we compute, for every pixel of I_2 the minimum Euclidean norm of the intensity difference to each pixel of $I_{1\to 2}$ in a neighborhood $\mathcal{N}_{i,j}$ around the projected pixel. We compute the size of this neighborhood by propagating the pose uncertainty, obtained while recording the images, into the image points (see Section 5.1.1). In detail, we search within the 3σ area given by $\mathbf{\Sigma}$ and select the pixel with the smallest difference:

$$D_{1\rightarrow2}(i,j) = \min_{k,l\in\mathcal{N}_{i,j}} ||I_2(i,j) - I_{1\rightarrow2}(k,l)||_2, \tag{5.4}$$

where i, j, k, l are pixel coordinates and the neighborhood $\mathcal{N}_{i,j}$ is defined as:

$$\mathcal{N}_{i,j} = \left\{ \forall(k,l) \in I_{1\rightarrow2} \;\middle|\; \begin{bmatrix} i-k \\ j-l \end{bmatrix}^{\mathsf{T}} \mathbf{\Sigma}^{-1} \begin{bmatrix} i-k \\ j-l \end{bmatrix} < d^2 \right\}, \tag{5.5}$$

where $d^2 = 11.82$ is the critical value of the χ^2_2 distribution corresponding to a probability of 99.73%, i.e., the 3σ boundary on the normal distribution. Finally, we normalize $D_{1\rightarrow2}$ to values between $[0,1]$. Figure 5.2d shows the result of this procedure.

If there is no change in the 3D model between the acquisition time and the time when the images have been taken, all pixels in I_1 should correctly re-project onto I_2. Therefore, I_2 and $I_{1\rightarrow2}$ should be identical and $D_{1\rightarrow2}$ should be small or equal to 0 for each pixel. If there is, however, a change in the model, pixels corresponding to the change re-project onto the wrong place in I_2. Thus, $D_{1\rightarrow2}$ allows us to identify the changes, as long as not all pixels in the current images have the same RGB value, i.e., represent a large homogeneous area.

The process, however, has ambiguities. As Figure 5.3 illustrates, a single point $\mathbf{X}_c$ corresponding to a change in the 3D model generates two pixel locations, $\mathbf{x}_{1\rightarrow2}$ and $\mathbf{x}'_{1\rightarrow2}$, in $D_{1\rightarrow2}$, one corresponding to the change in I_1 re-projected onto I_2 and one corresponding to the change in I_2 re-projected onto I_1. To eliminate this ambiguity, we use multiple pair-wise image comparisons as described in the following section.

5.1.3 Inconsistency Detection using Multiple Images

The ambiguity produced by the re-projection of an image onto another one can be eliminated by considering multiple image pairs. Figure 5.4 shows how a pixel belonging to the same change in a third image I_3 re-projects onto I_2 at two different locations. It is important to note that one of the two points is mapped to the same location as a change detected by re-projecting I_1 onto I_2. Thus, the pixels that re-project onto the same region of I_2 from the other images represent the real change.

To localize the changes, we therefore compare an image with its m neighboring keyframe images. For each image I_t, we store an inconsistency image D_t resulting from the pixel-wise minimum over all the inconsistency images obtained from the

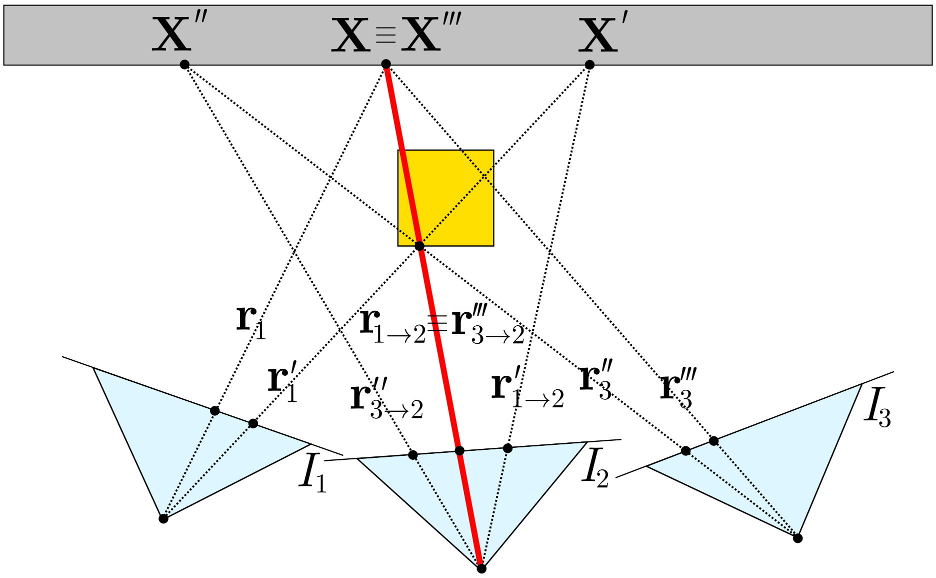


Figure 5.4: Ambiguity elimination using multiple images. When re-projecting I_1 and I_3 onto I_2, only one ray (therefore one pixel) is coincident. The thick red line represents that coincident ray.

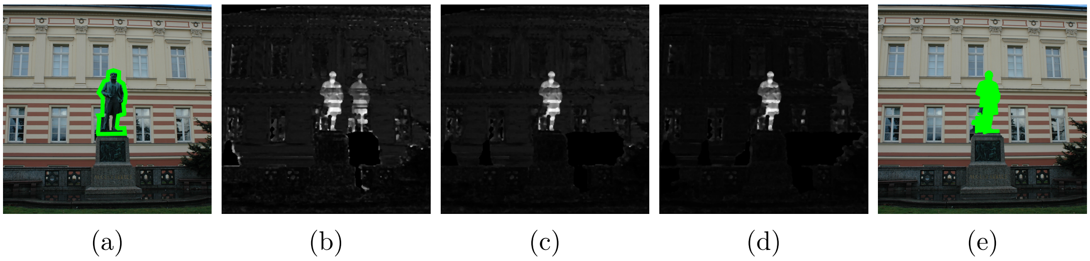


Figure 5.5: (a) The statue (here manually marked in green) is not in the model. (b) Inconsistencies between 2 images ($m = 1$). (c) Inconsistencies between 3 images ($m = 2$). (d) Inconsistencies between 4 images ($m = 3$). (e) Original image masked with the segmented area obtained from the inconsistency image with $m = 3$.

neighboring images re-projected onto I_t:

$$D_t(i,j) \quad = \quad \min\{D_{s\to t}(i,j),\ \forall s \in \mathcal{S}(t)\}, \tag{5.6}$$

where $\mathcal{S}(t)$ is the set of m neighboring keyframe images of I_t. In our implementation, we typically use the four closest images in time to I_t. Note that, for the algorithm to work, the baseline between the images must be sufficient, therefore the images must be recorded with a fixed minimum distance between each other. Figure 5.5 depicts the output of Equation (5.6), for $m = 1, 2, \text{and}\, 3$. Even though it is not easy to see in Figure 5.5, the noise in the total inconsistency image is substantially smaller when using more than $m = 2$ neighboring images. Thus, we stick to $m = 4$, although using $m = 2$ is theoretically sufficient. This means that we use a sequence of 5 keyframe images for the change detection.

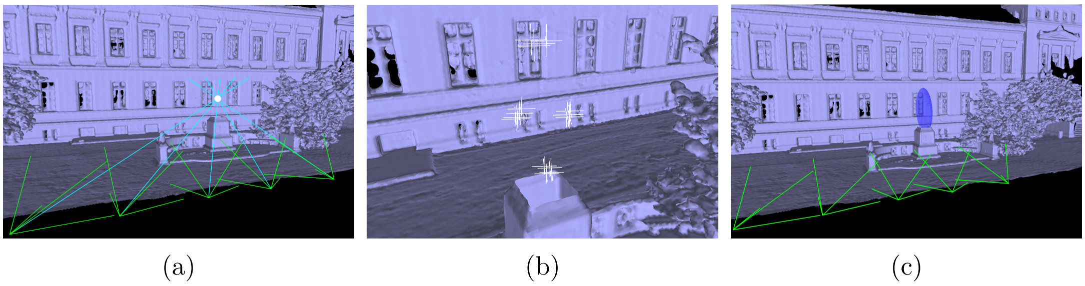


Figure 5.6: (a) Example triangulation with five images. The white lines are the back-projected rays and the white point represent the triangulated point. (b) Sigma points projected in 3D. (c) The result of our algorithm, i.e. the 3D region where the change is.

5.1.4 Segmentation and Data Association

The procedure explained so far enables us to identify the pixels in each image where changes occur. For reliably computing the regions of change, we first filter out the noise with an erosion-dilation procedure, then apply a standard border following algorithm [127]. We discard all the regions with a contour shorter than a threshold (in our implementation 50 px for images with horizontal resolution of 500 px) to filter out noise and changes that are too small. The next step is to associate the regions from the images with each other. To do that, we compute and compare hue-saturation histograms region-wise and perform standard cross-correlation together with a simple geometric consistency check using the epipolar lines.

5.1.5 Estimating the Location of Change

Once we obtain the segmented 2D regions and the association between them, we proceed to estimate the 3D location of the change. To simplify the notation in the remainder of this section, the following equations will refer to a single change in images, i.e., dropping an index referring to individual regions. The whole procedure is repeated for every region of detected change.

To estimate the 3D volumes in which the changes occur, we first compute, for every region identified as a change, the mean location $\bar{\mathbf{x}}_t$ and spread in form of the covariance $\mathbf{\Sigma}_t$ in the image. We then compute, for each change, a 3D point $\overline{\mathbf{X}}$ in the 3D world coordinates by triangulating the mean location in each image [42]. Specifically, we setup a system of equations in the form:

$$\mathbf{A}\overline{\mathbf{X}} = \mathbf{0}, \quad \text{with} \quad \mathbf{A} = \begin{bmatrix} \mathsf{S}(\bar{\mathbf{x}}_1)\mathbf{P}_1 \\ \vdots \\ \mathsf{S}(\bar{\mathbf{x}}_n)\mathbf{P}_n \end{bmatrix}, \tag{5.7}$$

where $\mathbf{A}$ is a $3n \times 4$ matrix composed by 3×4 blocks, n is the number of images,

$\mathbf{P}_t$ is the projection matrix relative to image I_t, and $\mathsf{S}(\bar{\mathbf{x}}_t)$ is the skew symmetric matrix corresponding to the mean pixel $\bar{\mathbf{x}}_t$, in homogeneous coordinates, i.e.:

$$\bar{\mathbf{x}}_t = \begin{bmatrix} x_t \\ y_t \\ w_t \end{bmatrix}, \quad \mathsf{S}(\bar{\mathbf{x}}_t) = \begin{bmatrix} 0 & -w_t & y_t \\ w_t & 0 & -x_t \\ -y_t & x_t & 0 \end{bmatrix}. \tag{5.8}$$

We solve this system using singular value decomposition and retrieve $\overline{\mathbf{X}}$ by taking the right-singular vector of $\mathbf{A}$ belonging to its smallest singular value (see Figure 5.6a for an example of triangulation). For each change in the image, we additionally compute the K sigma points $\mathbf{v}_t^{(k)}$ $(k = 1 \dots K)$ corresponding to $\bar{\mathbf{x}}_t$ and $\mathbf{\Sigma}_t$ and project the sigma points to the 3D space to estimate the region of change in 3D. Using the sigma points allows for a better propagation of a Gaussian through a non-linear function than first-order error propagation, see [58] for details. To compute the 3D position of the sigma points, we define for each image a plane $\widehat{A}_t$ passing through $\overline{\mathbf{X}}$ with normal equal to the direction of the ray $\bar{\mathbf{r}}_t$ obtained through Equation (5.2) for $\bar{\mathbf{x}}_t$.

We define the plane in homogeneous coordinates as a 4-dimensional vector:

$$\widehat{A}_t = \begin{bmatrix} \bar{\mathbf{r}}_t \\ d \end{bmatrix}, \tag{5.9}$$

where the last element $d = \bar{\mathbf{r}}_t^{\mathsf{T}} \overline{\mathbf{X}}$ is the distance between the camera and $\overline{\mathbf{X}}$. The projection of $\mathbf{v}_t^{(k)}$ on $\widehat{A}_t$ is the intersection $\mathbf{V}_t^{(k)}$ between the plane and the ray $\mathbf{r}_t^{(k)}$ generated from $\mathbf{v}_t^{(k)}$. We compute $\mathbf{V}_t^{(k)}$ by expressing $\mathbf{r}_t^{(k)}$ in Plücker coordinates as a line $\mathbf{L}_t^{(k)}$ joining the camera projection center $\mathbf{C}_t$ and a point $\mathbf{Y} = \mathbf{C}_t + \mathbf{r}_t^{(k)}$ along the ray:

$$\mathbf{L}_t^{(k)} = \begin{bmatrix} \mathbf{L}_h \\ \mathbf{L}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{C}_t - \mathbf{Y} \\ \mathbf{C}_t \times \mathbf{Y} \end{bmatrix}, \tag{5.10}$$

where $\mathbf{C}_t$ and $\mathbf{Y}$ are expressed in inhomogeneous coordinates. From $\mathbf{L}_t^{(k)}$, we compute the transposed Plücker matrix:

$$\mathbf{\Gamma}^{\mathsf{T}}(\mathbf{L}_t^{(k)}) \quad = \quad \begin{bmatrix} \mathsf{S}(\mathbf{L}_0) & \mathbf{L}_h \\ -\mathbf{L}_h^{\mathsf{T}} & 0 \end{bmatrix}, \tag{5.11}$$

where $\mathsf{S}(\mathbf{L}_0)$ is the skew symmetric matrix corresponding to $\mathbf{L}_0$. Finally, we obtain $\mathbf{V}_t^{(k)}$ as:

$$\mathbf{V}_t^{(k)} \quad = \quad \mathbf{\Gamma}^{\mathsf{T}}(\mathbf{L}_t^{(k)}) \hat{A}_t. \tag{5.12}$$

We repeat this procedure for the sigma points from each mean and covariance matrix of the same region in every image. In this way, we can quickly estimate the approximate 3D location of the change without computing a dense reconstruction of the scene, see Figure 5.6b. The mean and the covariance of the position of these points represent the 3D area where the change occurs, see Figure 5.6c.

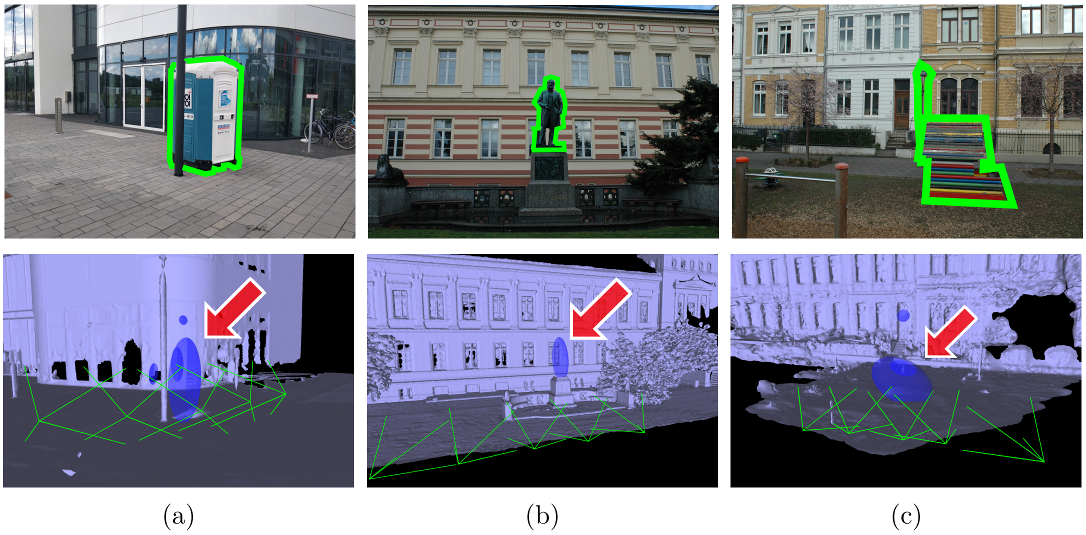


Figure 5.7: Results of our experiments on three different outdoor datasets. For each dataset, the top image shows the changes (here manually marked in green), while the bottom image shows the 3D region, identified by our algorithm, where the changes are.

5.2 Experimental Evaluation

The focus of this work is a comparably fast approach to identify changes in a given 3D model using a sequence of new images. Thus, our experiments are designed to show the performance of our approach and to support the two claims that we made in the beginning of the chapter, i.e., our method (i) can localize changes in the environment using a 3D model obtained in the past and a sequence of new keyframe images, and (ii) can be executed fast enough to run on an exploring robot, i.e., the average execution time should be in the order in which the sequence is recorded, here in the order of a few seconds for around five keyframe images.

We perform the evaluations on own datasets, as well as on a subset of the ScanNet dataset provided by Dai *et al.* [26]. Furthermore, we use the dataset provided by Taneja *et al.* [130] to provide a comparison with their method. Throughout all the experiments, we use a sequence of $n = 5$ images and for each image of the sequence, we compute the inconsistencies with $m = 4$ neighboring images, i.e., for these sequences all the neighboring images. Before the execution of the algorithm, we resize every image to a fixed width of 500 pixels.

5.2.1 Qualitative Evaluation

The first experiment is designed to illustrate the capability of our approach to localize a change in 3D given a model and a small sequence of images. Figure 5.7 depicts the results of the algorithm on three different outdoor datasets, while Figure 5.8 shows the results on three indoor scenes from the ScanNet dataset [26].

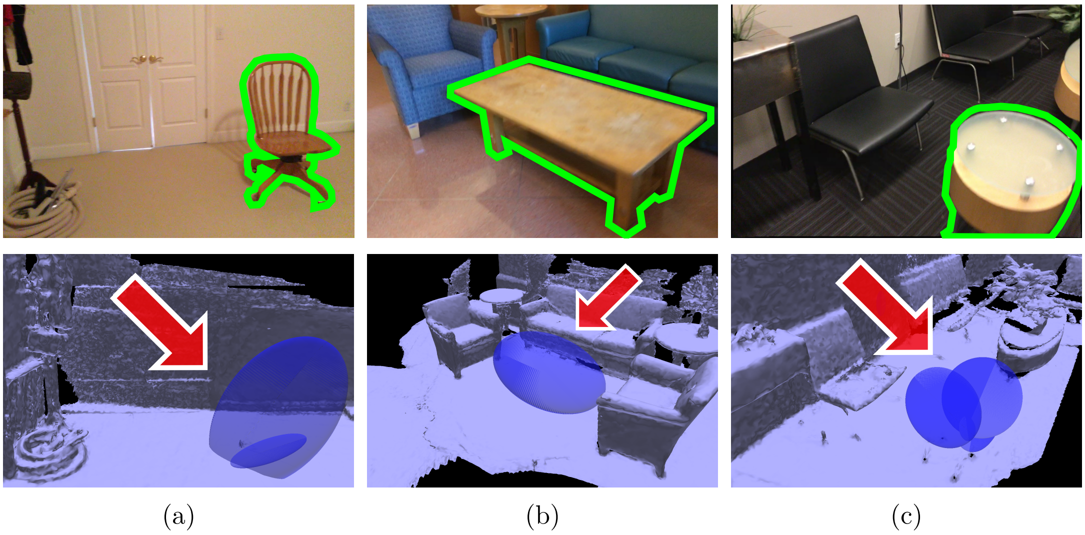


Figure 5.8: Results of our experiments on three different indoor scenes from the ScanNet dataset [26]. For each dataset, the top image shows the changes (here manually marked in green), while the bottom image shows the 3D region, identified by our algorithm, where the changes are.

In all our tests, the localized 3D regions reflect the actual position of the changes. This information can allow an exploring or mapping robot to inspect the changed regions in more detail and collect more observations to update the previously built model.

5.2.2 Quantitative Evaluation

To provide a quantitative evaluation, we project the 3D results of our approach onto the original 2D images in the sequence and we compare the 2D projection to a manually labeled ground truth. To get the final score for a dataset, we compute

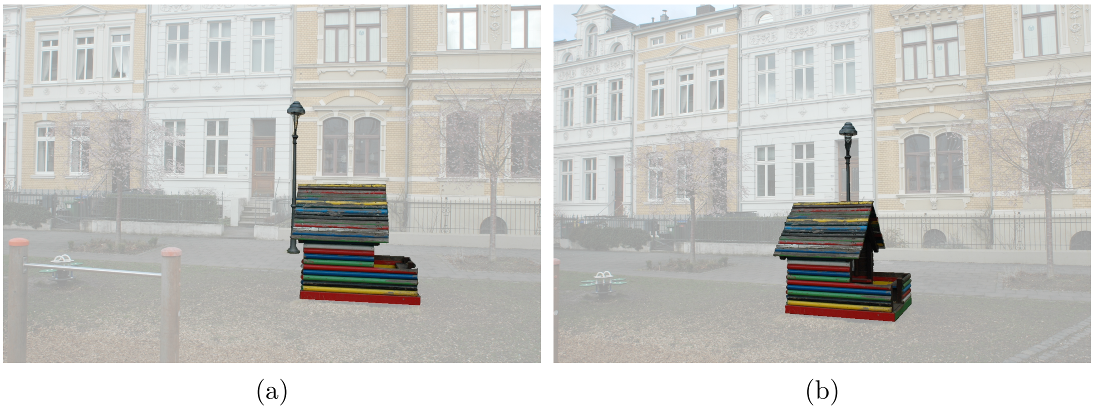


Figure 5.9: (a) The house does not occlude the lamp. (b) The lamp is occluded: it is impossible to guess its bounding box. Background modified for better visibility.

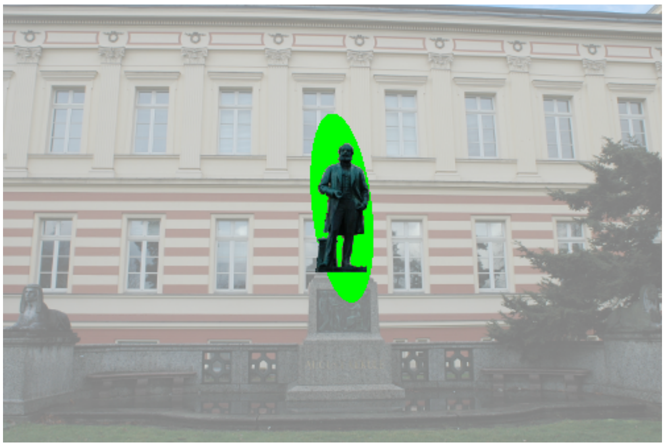


Figure 5.10: Result of our algorithm projected on the original image. The statue (manually segmented from the image) is not present in the model. The green ellipse is the projection of the result of our algorithm on the image. Background modified for better visibility.

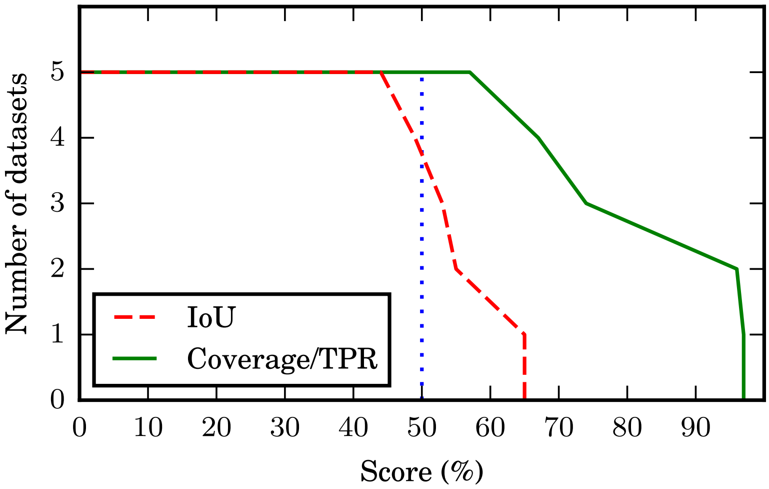


Figure 5.11: Evaluation on our outdoor datasets. The plot represents the number of datasets on which our approach achieved a score above a certain threshold.

the average score among all the images in the sequence. We use two different evaluation criteria. The first one is the *intersection over union* (IoU), which is one of the most commonly used metrics for image segmentation [36]. Formally, the IoU is computed by considering the intersecting area between the evaluated and the ground-truth regions and divide it by the area of the union of the two regions. The common approach is to compute the IoU using bounding boxes. However, since we are evaluating the changes projected in a sequence of images, occlusions may happen between multiple changes and that makes it impossible to evaluate the bounding box of a single change, see Figure 5.9 for an example. Therefore, we compute the intersection over union directly between the segmented ground truth and the projected ellipsoid used to indicate the change in our approach. As Figure 5.10 shows, the intersection over union is not always representative of

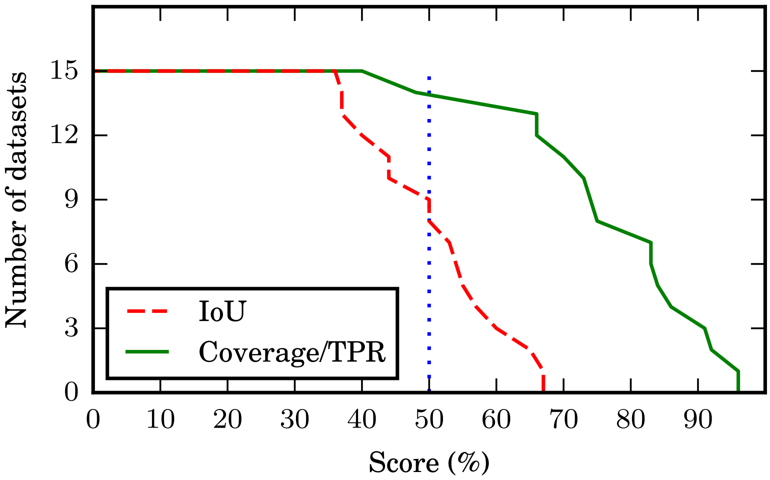


Figure 5.12: Evaluation on the indoor ScanNet datasets. The plot represents the number of datasets on which our approach achieved a score above a certain threshold.

the quality of our detection. In this case, the algorithm successfully identified the change, but the IoU score is only 44%. This results from the fact that we correctly identify the location of change, but have a substantial discrepancy between the two shapes, as we only compute an ellipsoid. Thus, we additionally provide the measurement of *coverage* (or True Positive Rate) of our results, i.e. the intersecting area between our detection and the ground truth, over the full area of the ground truth. This allows us to measure whether the change is properly covered or not. In the case of Figure 5.10 the coverage score is 97%, meaning that the algorithm is able to detect the change appropriately.

To test our algorithm, we recorded five outdoor datasets. To also consider other datasets, we additionally used the ScanNet dataset by Dai *et al.* [26], which provides indoor scenes recorded with an RGB-D camera. Here, we selected 15 scenes, we removed some objects from the 3D models and run our algorithm on a sequence of RGB images containing the missing objects. Figure 5.11 and Figure 5.12 show the number of datasets (on the y axis) over a certain threshold (on the x axis). The blue dotted line is the 50% threshold, which is the one commonly used in the image segmentation literature [36]. On all our datasets and on 12 out of 15 ScanNet datasets, the IoU score is over 40%. This shows that our algorithm is generally able to detect the position of the changes, although with a significant approximation in terms of shape (given that we only compute the ellipsoid and not the exact 3D structure). The coverage score, considerably higher, further supports this claim.

Table 5.1: Average execution time for different datasets.

Dataset	Average execution time [s]
Outdoor data	1.64±0.19
ScanNet (indoor)	1.95±1.08
All	1.88±0.94

Table 5.2: Results for the datasets by Taneja *et al.*

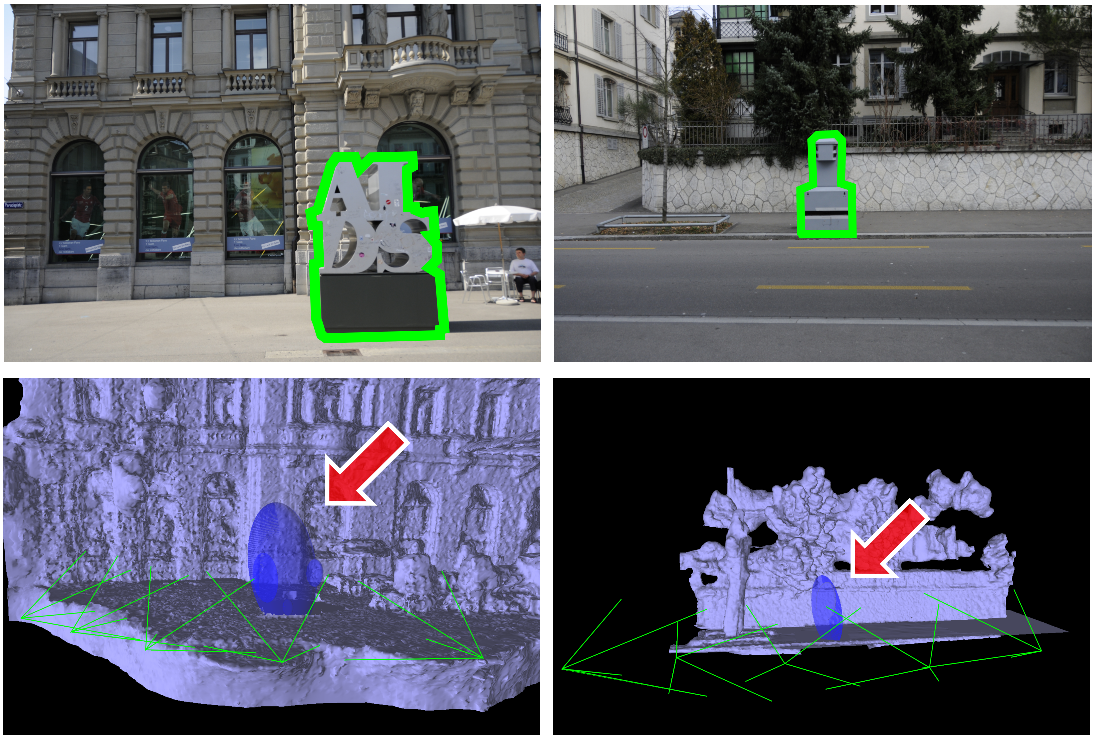
	Structure	Speedcam
IoU	51%	52%
Coverage	86%	88%
Time [s]	1.43	1.13

5.2.3 Execution Time

The next experiment is designed to support the claim that our approach runs fast enough for processing on an exploring robot. We therefore measured the execution time of our approach on a common, lightweight laptop with an Intel Core i7 processor and an embedded Intel GPU. All the operation were executed on a single core of the CPU except for the re-projection operations, which are implemented in OpenGL and therefore executed by the integrated GPU of the Intel processor. Table 5.1 shows the average execution time needed to process sequences of five images from different datasets as well as the standard deviation. The numbers support our second claim, namely that the computations can be executed fast enough for operation on an exploring robot. On all datasets, the whole process takes less than 2 s, which is shorter than the time needed to record the five keyframe images. Even though the process is clearly not real-time in a strict sense, it is fast enough to be executed on a real robot at a low frequency to trigger exploration or additonal mapping actions.

5.2.4 Comparison to the Approach by Taneja et al.

Finally, we want to briefly compare our results with those obtained by Taneja *et al.* [130]. The comparison is done based on two of the datasets that they provide and report on. We chose the "Speedcam" dataset, as it is the only one for which the authors provide the ground truth, and the "Structure" dataset, as it is the one that appears more often in their paper. We created the ground truth for the second dataset by segmenting the pictures manually, as it is not provided by the authors. Figure 5.13 and Table 5.2 illustrate the results of our algorithm on the datasets. Their approach uses a computationally expensive graph cut labeling on



(a) Structure

(b) Speedcam

Figure 5.13: Results of our approach on the datasets by Taneja *et al.* For each dataset, the top image shows the changes (here manually marked in green), while the bottom image shows the 3D region, identified by our algorithm, where the changes are.

a 3D voxelization of the scene. Their method typically provides a more accurate estimate of the region of change (in the order of $25{\times}25{\times}25\,\text{cm}^3$ voxels) than our estimate using the mean and covariance.

The disadvantage of their method, however, is the computational demands as they require computation times in the order of 1 min per region (reported by the authors, see [130]), whereas we can process the same datasets in about 1 second. Thus, for most robotics applications, where an online feedback is expected, our approach is better suited.

To summarize, our evaluation shows that our method can estimate the 3D location of changes in the environment. At the same time, the algorithm is fast enough to be used by an exploring robot to focus on the areas that have changed.

5.3 Related Work

Building 3D models can be an expensive process as it requires a good coverage of the environment and potentially dedicated sensors or equipment. To reduce this cost, it is important to identify, on an existing model, the parts that have changed, and direct the exploration towards those locations. For this reason, 3D change detection is an increasingly popular topic, see [95].

In the past, many 2D change detection algorithms have been proposed [97]. Several of such methods are affected by lighting conditions, seasonal changes, weather conditions, and other differences that may occur between the recording of the old and the new images. However, under certain conditions, the 2D approaches can still be useful, e.g. for monitoring a tunnel surface, as proposed by Stent *et al.* [122]. Another limitation of 2D approaches is that the images often do not provide information on the actual 3D location of the change. Sakurada *et al.* [105] try to overcome these problems by estimating the probabilistic density of the depth from the old set of images and by comparing it with the depth computed from the new set of images. Eden *et al.* [34] compare 3D lines in the images instead of using color or intensity information. A more recent approach by Sakurada and Okatani [104] instead uses a deep convolutional neural network to detect changes in omnidirectional images. Alcantarilla *et al.* [1] also use a deep convolutional neural network, but they additionally combine it with a dense reconstruction technique. A different approach is to formulate the problem of change detection as an optimal image labeling problem in the Markov Random Field framework, as proposed by Kovsecka [65], to detect different kind of changes between images of street scenes.

Change detection is a very popular topic in the remote sensing community [53]. Heller *et al.* [51] propose a framework for change detection using satellite and aerial images, by building digital elevation models and comparing 3D geometries. Crispell *et al.* [24] also focus on aerial and satellite imagery and introduce a variable-resolution probabilistic 3D model based on an octree to enable change detection. Malpica *et al.* [72], instead, combine satellite imagery and laser scanner data for updating buildings for a vector geospatial database. The buildings are detected using support vector machines. Dini *et al.* [30] obtain the depth information using high-resolution stereoscopic satellite images and detect changes against a GIS database, by creating digital surface models, detecting height changes, and filtering the result to remove noise. Chen *et al.* [19], in contrast, focus solely on aerial RGB-D images from a UAV. They detect changes by computing a depth difference map and a grayscale difference map and using random forest classification and component connectivity analysis techniques to segment the changes out.

Another approach to 3D change detection is to build a 3D model from the new images through Multi-View Stereo and then compare the new model with the old one. However, this is often a rather time consuming activity. Golparvar-Fard *et al.* [46] use this approach combined with a support vector machine classifier to obtain an updated voxelized model of the environment.

If a depth sensor is available, the change detection can be performed directly on the pointcloud data. An example of this approach is the work of Girardeau *et al.* [45], which detects changes between two ground LiDAR scans using octrees. A similar approach was used more recently by Kang *et al.* [61]. They focus on detecting disappearing changes in building models in the form of terrestrial point clouds, using an approach based on the Hausdorff distance. Andreasson *et al.* [5] also employ pointclouds obtained from a laser scanner, but they additionally integrate color information from a camera. They store the model using a normal distribution transform representation and perform change detection by computing a spatial difference probability and a color difference probability between a new point and the reference model. Choi *et al.* [21] detect changes between two LiDAR datasets by performing a subtraction between digital surface models and by classifying surface patches into pre-defined classes. Xiao *et al.* [141] use an alternative representation that is occupancy-based, but stores scanning rays and local point distributions rather than voxels. The change detection is performed by combining the advantages of the used representation and a point-to-triangle distance-based method. Finally, another possibility of comparing pointclouds is to compact the data using a Gaussian Mixture Model and finding the consistent matching in the Gaussian Mixture Model feature space, as proposed by Nunez *et al.* [83].

With the advent of cheap RGB-D cameras, a problem that has become popular in the robotics community is to map the static part of the environment, while segmenting out the dynamic part. This is mostly achieved by detecting changes between consecutive frames and identifying which objects are movable and which are static. An example of such approach is the work by Finman *et al.* [39], which detect changes in RGB-D maps by computing the 3D-difference of the models and use such changes to train segmentation algorithms. A similar approach is the one by Ambruş *et al.* [4] which performs change detection between pointclouds by identifying clusters of points that represent the change, and the one by Fehr *et al.* [38], which is based on a TSDF representation and use the detected changes to build a database of dynamic objects.

If a previously built 3D model is available but no updated depth information, a popular and effective approach is to infer the changes of the environment using the 3D model and a sequence of newly acquired images. One way to achieve this is to maintain a voxelized model of the environment and detect the probability of change in it by comparing the color of a voxel and the color of the pixels in the images onto which it projects. Examples of this approach are the one by Ulusoy and Mundy [134] or the one by Pollard *et al.* [92].

Another relevant strategy that uses an existing 3D model and newly acquired images is to identify changes by re-projecting images onto each other by passing through the existing model and compare the inconsistencies in the re-projection.

Taneja *et al.* [130] use this technique on pairs of images, and apply a graph cut minimization to label the changed area in 3D in a voxelized model. This technique is also effective for large scale change detection [131]. In addition, Qin *et al.* [94] combine the pairwise detected inconsistencies by counting the rays that hit every pixel for each image, in order to get rid of the ambiguities. They stop at the image level and do not estimate the 3D location of the change. A similar strategy, proposed by Fehr *et al.* [37], is to project the current model into keyframes on a sparse grid of image coordinates and measure the number of samples whose re-projection rays intersects with the model.

In this chapter, we use a re-projection technique similar to [130] and [94] to identify the changed regions in the images. We resolve ambiguities by fusing multiple images and introduce a fast way for estimating the rough location of change in 3D. The whole process takes only a few seconds for an image sequence. In contrast to that, state-of-the-art approaches such as [130] or [134] have execution times in the order of minutes.

5.4 Conclusion

In this chapter, we presented a novel approach to identify geometric changes between the current state of the environment and a previously built 3D model using a short sequence of images. Our approach operates by identifying the changes in the images by re-projecting them onto each other, passing through the 3D model. We eliminate the ambiguities about possible changes by combining the inconsistencies from multiple pairs of images. We are then able to estimate the locations of changes in 3D and identify the changed region through a mean 3D point and a covariance matrix. The computational time of the whole process using multiple images is in the order of seconds. We implemented and evaluated our approach on different datasets. The experiments show that our method can correctly identify the changes in the environment with only five images and a total computational time of less than 2 s, which make the algorithm suitable for running on mobile robots.

Chapter 6

Simultaneous Localization and Mapping in Dynamic Environments

In Chapter 3, we described a SLAM algorithm that allows for estimating a 3D model of the environment using an RGB-D sensor in an online fashion. In Chapter 3, we assumed the environment to be static during the mapping process, the real world, however, contains dynamic elements. SLAM is especially challenging in dynamic environments, since moving objects may cause wrong correspondences, deteriorate the ability to estimate correct poses and hence corrupt the map. Thus, a robot needs to estimate, simultaneously to the mapping process, which parts of the environment are static or moving.

In this chapter, we propose *ReFusion*, a novel approach for dense indoor mapping that is robust to dynamic elements moving trough the scene while mapping. Our approach is completely geometric and does not rely on an explicit semantic interpretation of the scene. More specifically, we do not need to employ a deep neural network to detect specific dynamic classes, as other recent approaches do [102, 10]. In contrast to other recent purely geometric approaches [100, 109], we do not represent the model using surfels, but in the form of a truncated sign distance function (TSDF). This allows our technique to directly generate a high quality mesh of the environment. Moreover, the TSDF representation can be useful for planning, since it provides, by definition, the distance to the closest obstacle.

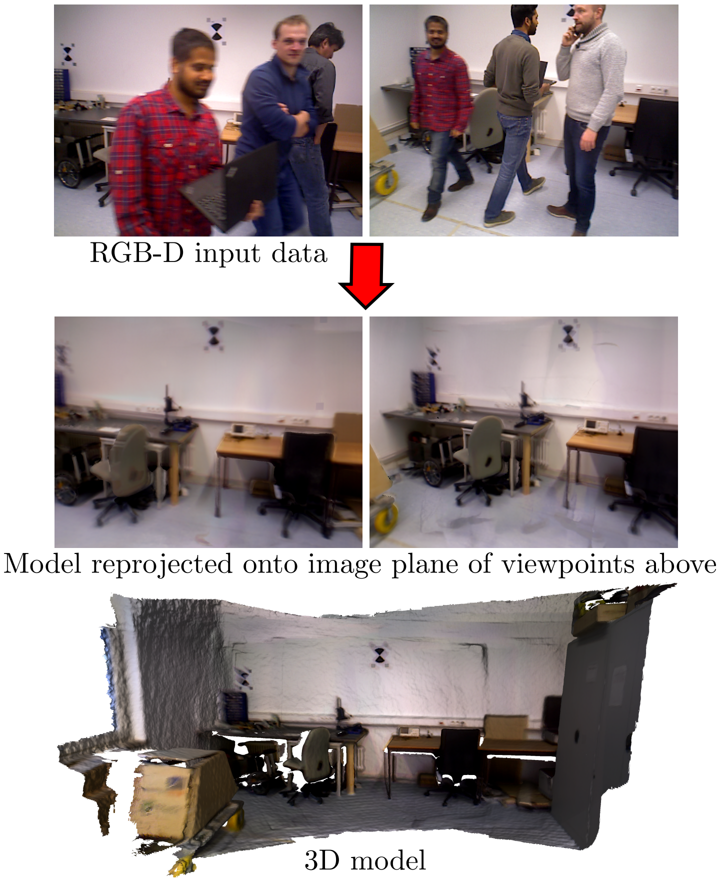


Figure 6.1: Result of our approach. Top: RGB frames from our dataset containing dynamics. Center: Reconstructed model without dynamics reprojected onto the image planes of the two frames from above. Bottom: Final mesh without the dynamics.

6.1 ReFusion: 3D Reconstruction in Dynamic Environments

The main contribution of this chapter is a novel and efficient SLAM algorithm, based on a TSDF representation, that is robust to dynamics via geometric filtering. It can be seen as an extension of our approach presented in Chapter 3. We propose to detect dynamics by exploiting the residuals obtained from the registration, in combination with the explicit representation of free space in the environment. This allows our approach to be class agnostic, i.e., it does not rely on a detector trained on specific categories of dynamic objects. Figure 6.1 illustrates the capabilities of our method. It shows two RGB frames from a dynamic scene and the resulting model built by our approach. As can be seen, all the dynamic objects are removed from the model. We release the implementation of our approach as open source software[1].

[1] https://github.com/PRBonn/refusion

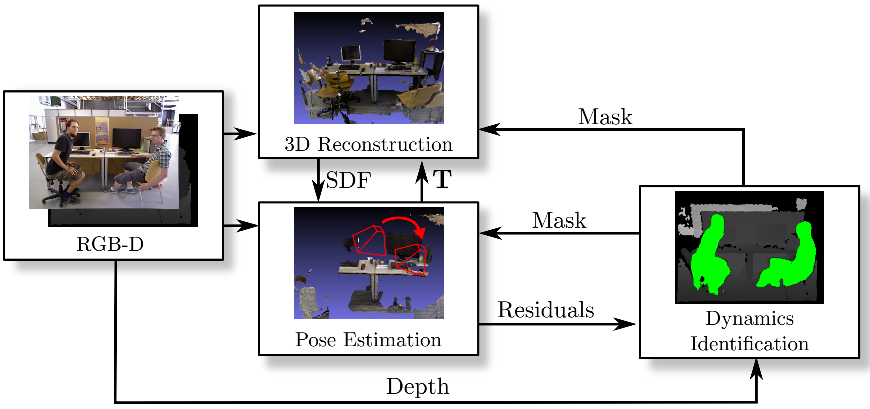


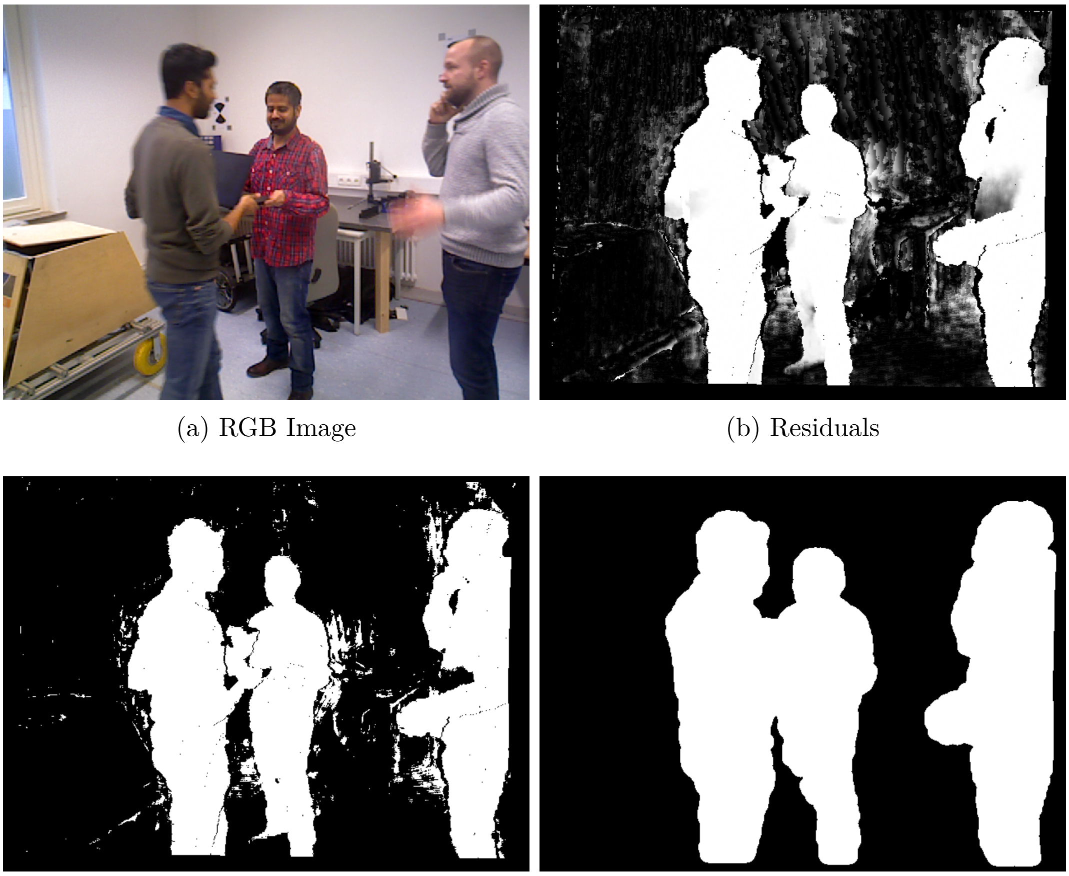
Figure 6.2: Overview of our approach. Given data from the RGB-D sensor, we first perform an initial pose estimation. Then, we use the obtained residuals, together with the depth information, to identify dynamic parts of the scene. The filtered images are then used to refine the pose **T** w.r.t. the TSDF given by our 3D reconstruction. With the updated sensor pose, we finally integrate the measurements into our 3D reconstruction of the environment.

We evaluate ReFusion on the TUM RGB-D dataset [125], as well as on our own dataset, showing the versatility and robustness of our approach, reaching in several scenes equal or better performance than other dense SLAM approaches. In addition, we publicly release our dataset[2], containing 24 highly-dynamic scenes recorded with an RGB-D sensor, together with ground truth trajectories obtained using a motion capture system. Furthermore, we provide a ground truth 3D model of the static parts of the environment in the form of a high resolution point cloud acquired with a terrestrial laser scanner. To the best of our knowledge, this is the first dataset containing dynamic scenes that also includes the ground truth model for the static part of the environment.

In sum, we make two key claims about our RGB-D mapping approach: it (i) is robust to dynamic elements in the environment and provides a camera tracking performance on par or better than state-of-the-art dense SLAM approaches, and (ii) provides a dense 3D model that contains only the static parts of the environment, which is more accurate than other state-of-the-art approaches when compared to the ground truth model.

Figure 6.2 illustrates the key processing steps of the proposed approach. Given the color and depth information of an RGB-D sensor, like Microsoft's Kinect, we first perform an initial pose estimation by exploiting directly the TSDF of our model representation explained in Chapter 3. By observing the residuals obtained from such registration, we detect the dynamic elements in the scene. With the filtered sensor information, where we discard regions containing dynamics, we further refine the pose of the sensor. With this refined estimated pose, we then integrate the sensor measurements, i.e., depth and color, into the model.

[2] `http://www.ipb.uni-bonn.de/data/rgbd-dynamic-dataset`



(a) RGB Image

(b) Residuals

(c) Raw mask

(d) Final mask

Figure 6.3: Steps of the mask creation. (a) Example RGB frame. (b) Residuals obtained from the registration. (c) Initial mask obtained from the residual. (d) Final refined mask after the floodfill algorithm.

6.1.1 Model Representation and Pose Estimation

The approach presented in this chapter is an extension of the SLAM technique presented in Chapter 3. Specifically, the two approaches share the same model representation based on TSDF and storage realized with voxel hashing. Moreover, both algorithms estimate the pose of the sensor using a point-to-implicit technique that exploits both geometric and color information. For further information about the basic SLAM algorithm, we refer to Chapter 3.

6.1.2 Dynamics Detection

To detect dynamic parts of the environment, we first perform an initial registration of the current RGB-D frame with respect to the model, as described in Section 3.1.2. Then, we compute for each pixel $\mathbf{p}_i$ its residual r_i w.r.t. the model as defined in Equation (3.4), i.e., the squared error of the distance of the point to the model. Figure 6.3b illustrates the residuals obtained from the registration

Algorithm 1: floodfill for generating $\mathcal{M}_\text{D}$.

```
Input: pixels of residual mask $\mathcal{M}_\text{R}$
Result: Mask $\mathcal{M}_\text{D}$
Let $\mathcal{Q}$ be a queue containing all the pixels to be masked.
Let $\mathcal{N}(\mathbf{p})$ be the set of neighbors of pixel $\mathbf{p}$
Add all pixels from $\mathcal{M}_\text{R}$ to queue $\mathcal{Q}$
while $\mathcal{Q} \neq \emptyset$ do
    Add all pixels inside $\mathcal{Q}$ to $\mathcal{M}_\text{D}$
    foreach $\mathbf{p} \in \mathcal{Q}$ do
        foreach $\mathbf{n} \in \mathcal{N}(\mathbf{p})$ do
            if $||D(\mathbf{p}) - D(\mathbf{n})|| < \theta \cdot D(\mathbf{p})$ then
                Add $\mathbf{n}$ to $\mathcal{Q}$ if $\mathbf{n} \notin \mathcal{M}_\text{D}$
            end
        end
        Remove $\mathbf{p}$ from $\mathcal{Q}$
    end
end
```

of the RGB-D frame depicted in Figure 6.3a. We select a threshold t as:

$$t = \gamma\tau^2, \tag{6.1}$$

where τ is the truncation distance used in our TSDF representation and γ is a value between 0 (everything is masked) and 1 (nothing is masked). Figure 6.4 shows a histogram of the residuals obtained after the initial registration of one image. The figure shows how most of the residuals concentrate below a certain value, except the ones belonging to dynamic parts of the environment. Every residual exceeding t contributes to the creation of a binary mask, as illustrated in Figure 6.3c. Such threshold-based segmentation is often not perfect and may fail to capture the whole dynamic object. Since we have depth information available, we use the eroded mask $\mathcal{M}_\text{R}$ to initialize a depth-aware flood fill algorithm, summarized in Algorithm 1, related to the region growing approaches used in [62, 102]. Given an initial region, defined by $\mathcal{M}_\text{R}$, the algorithm consists in adding neighboring pixels to that region as long as their depth does not differ more than a threshold θ. Finally, the mask is dilated again to cover eventual border pixels left out by the floodfill. Figure 6.3d shows the resulting mask after all the processing steps. We then perform a second registration without masked pixels and integrate the RGB-D information ignoring the masked pixels into the model using the newly obtained pose.

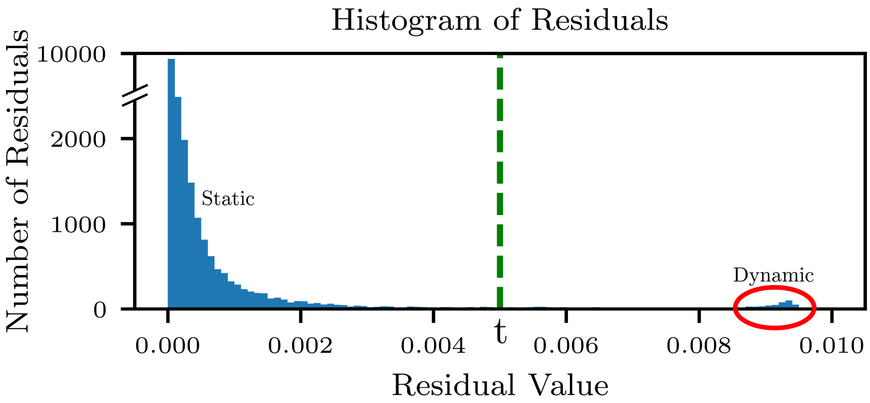


Figure 6.4: Histogram of residuals obtained after the registration of an image containing dynamic elements. The red ellipse highlights the residuals resulting from the dynamic parts. By considering only pixels with residuals under the threshold t, we can identify the dynamic parts of the image.

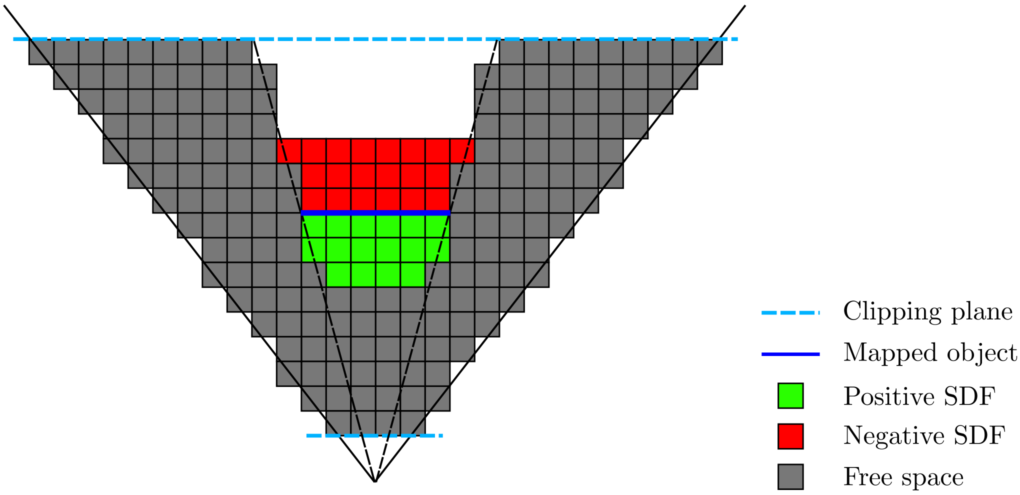


Figure 6.5: Explicit representation of free space. Every voxel inside the camera frustum that is not occluded and is outside the truncation distance is marked as free, see the grey cells in the figure.

6.1.3 Carving of Model and Free Space Management

One weakness of TSDF approaches is that they cannot keep track of perceived free space. However, being aware of the previously measured free space is a way to reject dynamic objects, as opposed to detecting them from their motion, which can be tricky given that the camera might also move. Assuming we would have free-space information, we can define a constraint for rejecting dynamics: a voxel that was found to be reliably empty can never contain a static object. Indeed, if a new range image points to a voxel being occupied, it can only be so because a dynamic object has entered it. Therefore, we can safely reject it. Doing so, we sidestep the notoriously difficult problem of tracking dynamic moving points, as done in [100, 102], but our technique only works if free-space information

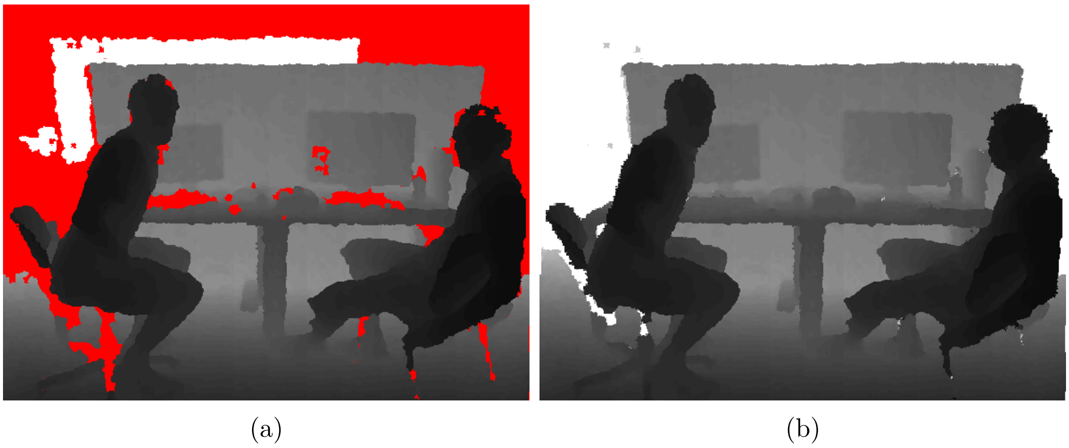


Figure 6.6: (a) Raw depth from the RGB-D sensor. Out-of-range values are highlighted in red. (b) Our virtual refined depth from 10 adjacent frames. (about 0.3 s delay)

is explicitly stored in the model. To integrate free-space information, we mark as free every non-occluded voxel in the camera frustum outside the truncation region (up to a clipping plane), see Figure 6.5 for an illustration. Note that this is particularly helpful in case of measurements of points that are too far from the sensor to be integrated into the TSDF. In our current implementation, we assign to free voxels an SDF value equal to the truncation distance τ. Doing this for every voxel in the frustum is suboptimal in terms of memory consumption. This may be improved by maintaining an octree-based representation of the free space.

Another consideration regarding free space is that if a previously static object moves, its voxels must be removed from the map. This corresponds to the fact that if a voxel previously mapped as static is detected as empty subsequently, this voxel used to contain a dynamic object and should, therefore, be marked as free. This behavior is automatically achieved by the TSDF representation, i.e., we update the SDF stored in the voxel by performing a weighted average between the current value and the truncation distance τ. Therefore, if a voxel is detected as free long enough, it will be reliably marked as free.

6.1.4 Handling Invalid Measurements

Marking free voxels in the camera frustum is effective as long as we know from the sensor that those regions of space are empty. However, common commercial RGB-D cameras return depth images that contain invalid measurements, i.e., pixels with a depth value of zero, as exemplarily shown by red pixels in Figure 6.6a. These measurements are invalid either because they are out-of-range (too close to the sensor or too far from it) or because they are not measurable, e.g., because of scattering media, reflecting surfaces, etc. For our approach to work in every possible case, it is necessary to distinguish between out-of-range and non-measurable values. Two methods to handle such cases are possible.

The first option is to make no distinction and not consider zero values, as it is commonly done in other RGB-D mapping approaches. In this case, our approach will work correctly when the whole scene is in the measurement range of the depth sensor. If there are out-of-range values, dynamic objects will be incorrectly added to the model, thus affecting its quality.

The second method is to "correct" non-measurable values if possible. To this end, we create a temporary model from n consecutive frames and generate virtual depths from the registered poses. Then, we fill in the original depth images by replacing every zero value with the corresponding value of the virtual depth. In this way, non-measurable values are usally reduced thanks to multiple observations from slightly different viewpoints. The remaining values are assumed to come from out-of-range measurements of the camera and are replaced with a high fixed depth value. Figure 6.6b shows an example of a virtual depth image obtained with this technique. This solution, however, assumes that nothing appears in front of the camera that is closer than the minimum range of the depth sensor.

In theory, this assumption can be relaxed in case we add an additional sensor, e.g., a simple sonar on top of the RGB-D sensor, that detects whether there are objects too close to the camera. Moreover, a disadvantage of this method is that the actual model is then generated with a delay of n frames, which might be suboptimal for some robotics applications.

In the following experiments, we employ the second option for modeling the sequences of the TUM RGB-D dataset, since the depth images contain out-of-range values. For the sequences of our dataset, we employ the first option, since the recorded depth is always within the valid range of the depth sensor.

6.2 Depth-Enhanced Neural Network-Based Dynamic Filtering

In contrast to our approach, described in Section 6.1, the current trend in the state of the art is to identify specific category of dynamic elements using a convolutional neural network (CNN), and remove them from the environment [10, 101]. To provide a comparison between our geometric approach and a CNN-based one, we implemented an algorithm based on our basic SLAM system (see Chapter 3) in combination with a CNN-based segmentation of people. We basically pre-process the RGB-D information using a CNN to remove people from the images before the registration, see Figure 6.7.

For the segmentation of people, we use an encoder-decoder CNN architec-

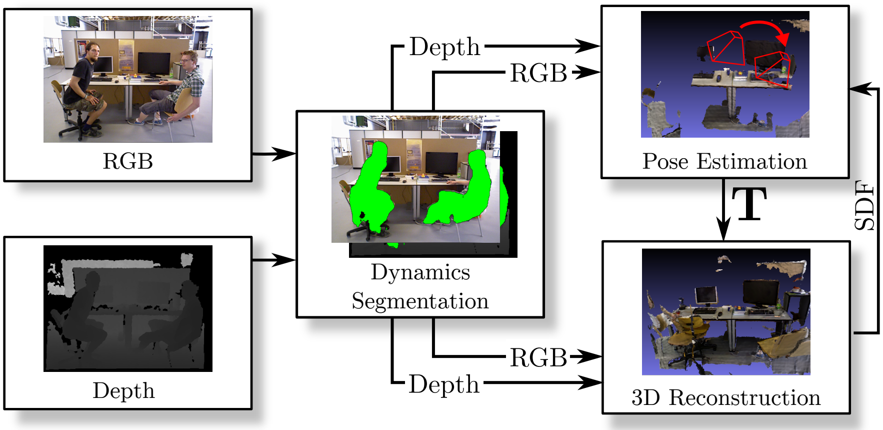


Figure 6.7: Overview of our CNN-based baseline approach. Given an image from the RGB-D sensor, we first compute the masked RGB and depth images with our dynamics segmentation to filter dynamics caused by people (depicted by green masks). The filtered images are then used to estimate the pose $\mathbf{T}$ w.r.t. the SDF given by our 3D reconstruction. With the updated pose of the sensor, we finally integrate the measurements into our 3D reconstruction of the environment.

ture [75] predicting pixel-wise labels, based on the inception architecture [128] and using residual connections [50]. The network was trained on the MS COCO dataset [69] people segments and the segmentation runs at 65 Hz using an image of size 640 × 480, and achieving a validation Jaccard index of 85%. Thus, the filtering can be performed at real-time speed, and further classes of dynamic objects can be added in a straight forward way. Such segmentation is not perfect and can fail to capture the whole body of the person. Therefore, we use the eroded RGB mask $\mathcal{M}_{\text{RGB}}$ to initialize a depth-based flood fill algorithm, in a similar way as described in Algorithm 1. Given the RGB-D mask $\mathcal{M}_{\text{D}}$, we found that a composition of a dilated $\mathcal{M}_{\text{RGB}}$ and $\mathcal{M}_{\text{D}}$ performs better then just using either one. We found that the flood fill otherwise misses parts that got eroded, but just the dilation of the RGB mask misses parts that can be extracted from the depth map. Using the computed masks, we perform the registration and fuse the data in the model as described in Chapter 3, ignoring the masked pixels.

6.3 Experimental Evaluation

The main contribution of this work is a TSDF-based mapping approach that is able to operate in environments with the presence of highly dynamic elements by relying solely on geometric information, i.e., our approach is completely class agnostic and does not require tracking of objects. Our experiments show the capabilities of our method and support our key claims, which are: (i) our approach is robust to dynamic elements in the environment and provides a camera track-

Table 6.1: Parameters of our approach in all experiments.

Parameter	Value
Voxel size	0.01 m
Truncation distance τ	0.1 m
Huber constant	0.02
Initial regularization parameter λ	0.002
Weight w_c	0.025
floodfill threshold θ	0.007
Residual threshold weight γ	0.5

ing performance on par or better than state-of-the-art dense SLAM approaches, and (ii) provides a dense 3D model that contains only the static parts of the environment, which is more accurate than other state-of-the-art approaches when compared to the ground truth model.

We provide comparisons with StaticFusion (SF) [109], DynaSLAM (DS) [10] and MaskFusion (MF) [102]. As our approach does not rely on deep neural networks, we make a distinction in our comparison between the pure geometric approach of DynaSLAM (G) and the combined deep neural network+geometric approach (N+G). Moreover, we provide the results of the SLAM system presented in Chapter 3, combined with a CNN (Section 6.2). We refer to such system as Base+CNN. We tested all approaches on the dynamic scenes of the TUM RGB-D dataset [125], as well as on our dataset, designed to contain highly dynamic scenes. We obtained the reported results by using the open source implementations available for the different approaches, with the exception of MaskFusion, where we only report results from the original paper [102].

In all experiments, we used the default parameters provided by the open source implementations of the approaches. For our approach, we used the set of parameters shown in Table 6.1, which we determine empirically, but similar values for those parameters gave comparable results.

In the presented tables, we separate the approaches that rely solely on geometric information, from approaches that rely also on neural networks. We highlight in bold the best result among the first category of approaches, as we focus mainly at class agnostic approaches.

6.3.1 Performance on the TUM RGB-D Dataset

The first experiment shows the performance of our approach with the TUM RGB-D dataset [125]. Note that, since the depth information from these sequences contains out-of-range values, we used the approach described in Section 6.1.4

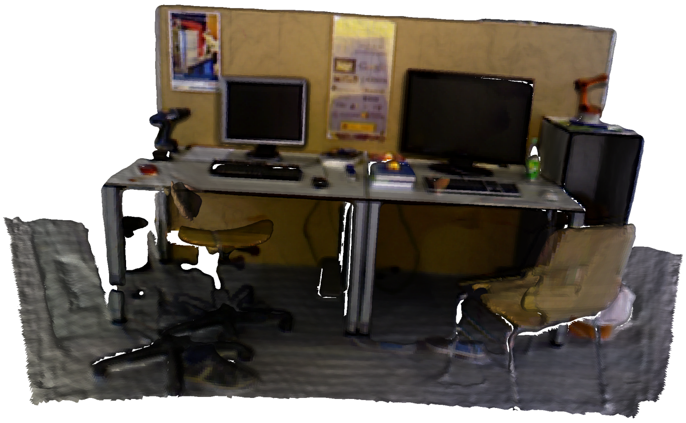


Figure 6.8: Final mesh obtained using our approach on the *walking_static* sequence, in which two people are continuously walking through the scene.

Table 6.2: Absolute Trajectory Error (RMS) [m] on dynamic scenes of TUM dataset.

	Purely geometric			Requires CNN		
	Ours	SF	DS (G)	DS (N+G)	MF	Base+CNN
Dense approach	✓	✓	✗	✗	✓	✓
sitting_static	**0.009**	0.014	**0.009**	0.007	0.021	0.008
sitting_xyz	0.040	0.039	**0.009**	0.015	0.031	0.063
sitting_halfsphere	0.110	0.041	**0.017**	0.028	0.052	0.181
walking_static	0.017	0.015	**0.014**	0.007	0.035	0.024
walking_xyz	0.099	0.093	**0.085**	0.017	0.104	0.065
walking_halfsphere	0.104	0.681	**0.084**	0.026	0.106	0.156
Max	0.110	0.681	**0.085**	0.028	0.106	0.181

to obtaine a refined depth, with the temporary model created from $n = 10$ consecutive frames, which corresponds to a model computation that is delayed by approximately 0.3 s, which should be acceptable for most setups.

Table 6.2 shows the results of all considered approaches on six sequences of the TUM dataset. From this table, it is clear that DynaSLAM outperforms the other methods. However, DynaSLAM is a feature-based approach and, in contrast to the other approaches, does not provide a dense model. The four dense mapping approaches show in most of the cases similar results, except for the sequence *walking_halfsphere*, where StaticFusion lost track due to the excess of dynamic elements at the beginning of the sequence, and the sequence *sitting_halfsphere*, where our approach and Basic+CNN show worse performance. In terms of 3D reconstruction, our approach is always able to create a consistent mesh of the

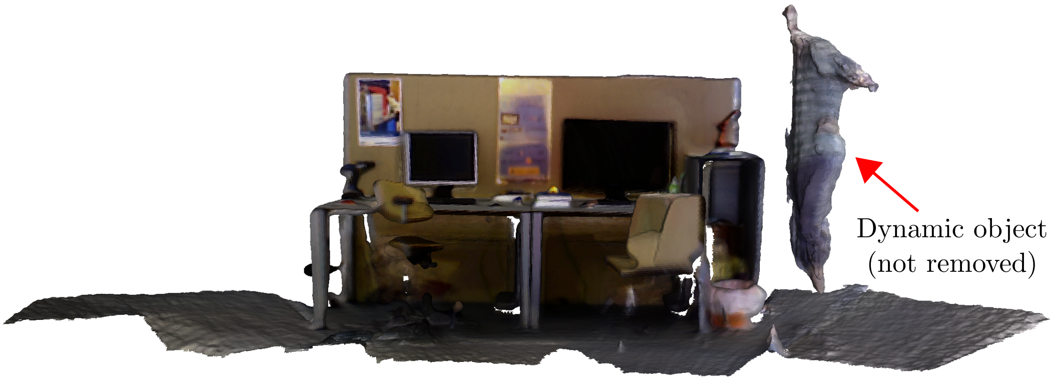


Figure 6.9: Final mesh obtained using our approach on the *walking_xyz* sequence. In this sequence, the camera follows the person on the right at the beginning and then never revisits that location. Therefore, the person is added in the model.

environment, see Figure 6.8 for an example.

The only case where the model built by our approach shows artifacts is on the *walking_xyz* sequence, where a person remains in the model, see Figure 6.9. This happens because the person is tracked by the camera at the beginning of the sequence and the location where the person stops is never revisited again. Therefore, the algorithm cannot know that the voxels in that location are actually free. This is confirmed by Figure 6.10, which shows the relative position error versus the elapsed time. It is evident from the figure that in the first four seconds, i.e., when the camera tracks the person, the error is particularly high. In sum, we are on par with state-of-the-art dense mapping approaches in terms of tracking, but we use a completely different technique based on TSDF instead of surfels. This allows for example to easily extract a detailed mesh of the environment, useful for many applications.

6.3.2 Performance on the Bonn RGB-D Dynamic Dataset

The second set of experiments consists of the comparison between the algorithms on our dataset. Our dataset includes 24 highly dynamic scenes, where people perform different tasks, such as manipulating boxes or playing with balloons, see Figure 6.11 for some example RGB frames. These tasks often obstruct the camera, creating particularly challenging situation for mapping approaches. We recorded the dataset using an ASUS Xtion Pro LIVE sensor, combined with an Optitrack Prime 13 motion capture system for the ground truth trajectories. Additionally a Leica BLK360 terrestrial laser scanner was used to obtain a ground truth 3D pointcloud of the static environment.

Table 6.3 shows the performance of different approaches on our scenes. The variety of sequences show interesting phenomena. For example, on the scenes where the dynamic component is a uniformly colored balloon, DynaSLAM out-

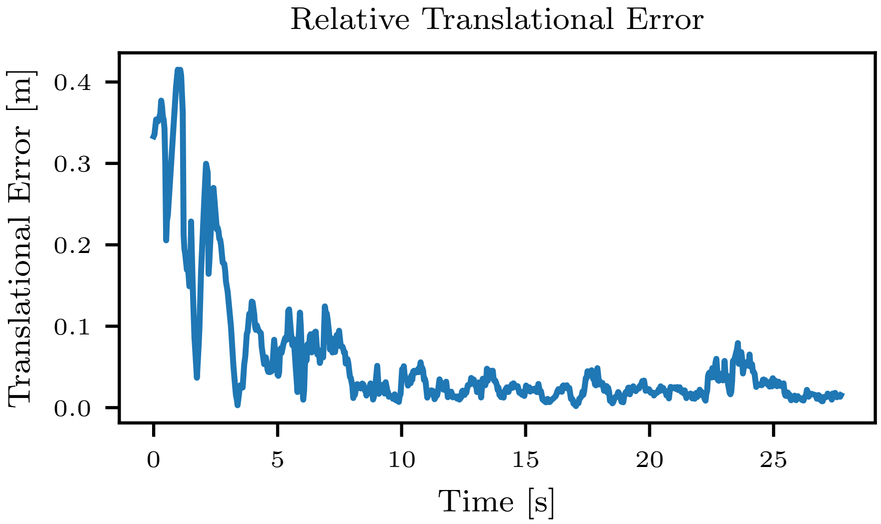


Figure 6.10: Relative translational error over time for the *walking_xyz* sequence. In this plot it is visible how the relative error is particularly high at the beginning of the sequence, when the camera is tracking the person. After the first four seconds, the error drops substantially.

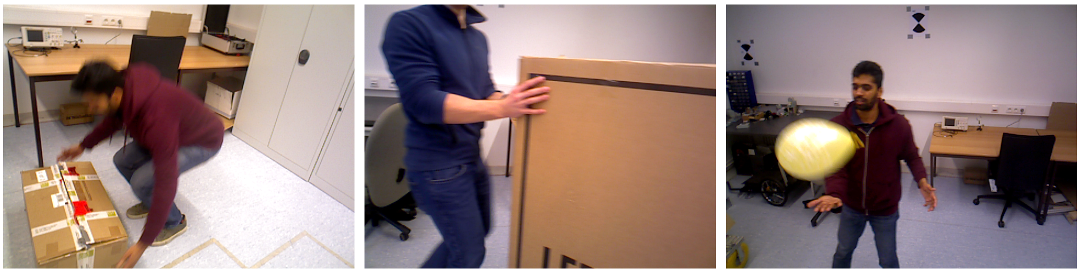


Figure 6.11: Example RGB frames from our highly dynamic dataset.

performs the dense approaches, because it cannot detect features on the balloon, which therefore does not affect the SLAM performance. Our approach performs best on scenes crowded with people, which are among the most challenging if no semantic segmentation algorithm is available. On sequences that involve the manipulation of boxes, the algorithms have mixed results, with our approach being better in about half of the cases and DynaSLAM being better on the other half. Note that DynaSLAM with the combined neural network and geometric approach performs the best in most cases. Moreover, our Base+CNN approach often outperforms our geometric approach. This is due to the heavy bias of having people in every sequence of our dataset, therefore the segmentation of people always helps the algorithm achieving better results. However, the worst performance of our approach is on par with the worst performance of DynaSLAM (N+G) and substantially better than the other approaches, showing that our approach is more robust to failure. This is a desirable quality when deploying robotic systems. Finally, as discussed in the previous section, our approach is able to build

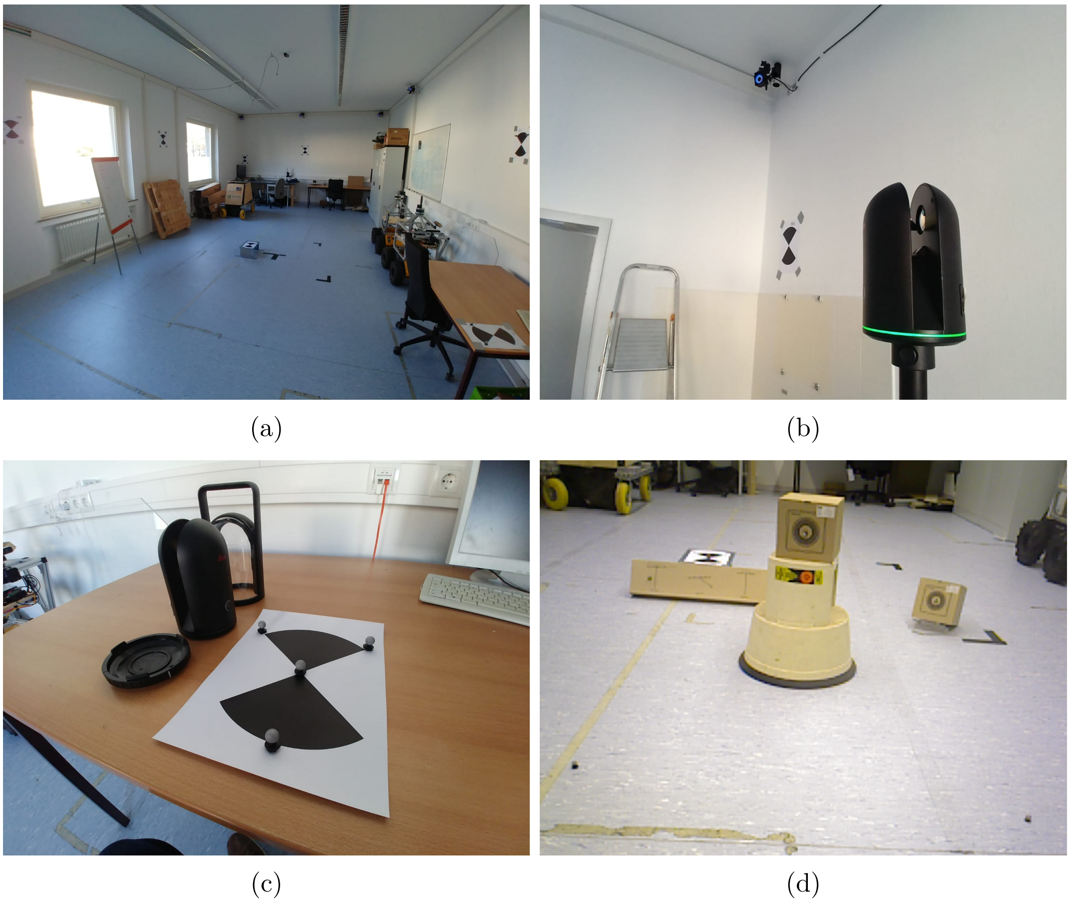
Table 6.3: Absolute Trajectory Error (RMS) [m] on our dataset. In this table, we shorten *obstructing_box* with *o_box* and *nonobstructing_box* with *no_box*.

	Purely geometric			Requires CNN	
	Ours	SF	DS (G)	DS (N+G)	Base+CNN
Dense approach	✓	✓	✗	✗	✓
balloon	0.175	0.233	**0.050**	0.030	0.142
balloon2	0.254	0.293	**0.142**	0.029	0.179
balloon_tracking	0.302	0.221	**0.156**	0.049	0.154
balloon_tracking2	0.322	0.366	**0.192**	0.035	0.286
crowd	**0.204**	3.586	1.065	0.016	0.110
crowd2	**0.155**	0.215	1.217	0.031	0.133
crowd3	**0.137**	0.168	0.835	0.038	0.255
kidnapping_box	0.148	0.336	**0.026**	0.029	0.101
kidnapping_box2	0.161	0.263	**0.033**	0.035	0.134
moving_no_box	**0.071**	0.141	0.317	0.232	0.056
moving_no_box2	0.179	0.364	**0.052**	0.039	0.134
moving_o_box	0.343	**0.331**	0.544	0.044	0.347
moving_o_box2	0.528	**0.309**	0.589	0.263	0.390
person_tracking	**0.289**	0.484	0.714	0.061	0.246
person_tracking2	**0.463**	0.626	0.817	0.078	0.348
placing_no_box	**0.106**	0.125	0.645	0.575	0.087
placing_no_box2	0.141	0.177	**0.027**	0.021	0.113
placing_no_box3	**0.174**	0.256	0.327	0.058	0.137
placing_o_box	0.571	0.330	**0.267**	0.255	0.818
removing_no_box	0.041	0.136	**0.016**	0.016	0.038
removing_no_box2	0.111	0.129	**0.022**	0.021	0.086
removing_o_box	**0.222**	0.334	0.362	0.291	1.065
synchronous	**0.441**	0.446	0.977	0.015	0.016
synchronous2	**0.022**	0.027	0.887	0.009	0.015
Max	**0.571**	3.586	1.217	0.575	1.065

a consistent model of the environment in most of the cases, see Figure 6.1 for an example of model from the scene *crowd3*.

6.3.3 Model Accuracy

The last set of experiments shows that our approach provides an accurate, dense 3D model that contains only the static parts of the environment. To perform



(a)

(b)

(c)

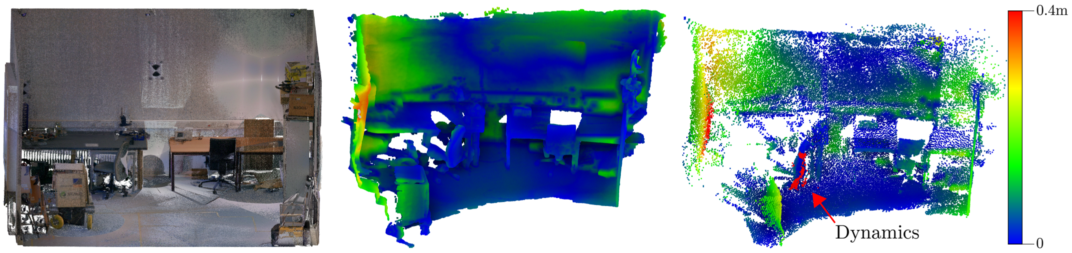
(d)

Figure 6.12: (a) Our test environment. (b) Terrestrial laser scanner. (c) Tilt and turn target. (d) Calibration setup used to align the sensor's reference frame with the motion capture system's one.

such experiments, we first built a high resolution point cloud of the static part of our test environment (Figure 6.12a) using a professional terrestrial laser scanner, the Leica BLK360 (Figure 6.12b). We then aligned the point cloud to our motion capture system's reference frame using tilt and turn targets (Figure 6.12c) that we located with both the laser scanner and the motion capture system. Figure 6.13a shows a section of our ground truth point cloud.

To align the model created by the algorithms to our ground truth, we transformed it from the reference frame of the RGB-D sensor, to the reference frame of the motion capture system. We aligned the two frames using the calibration setup shown in Figure 6.12d, where the markers positioned in the environment were known in both reference frames.

We compare the models built by our algorithm and by StaticFusion [109] for the sequences *crowd3* and *removing_nonobstructing_box* w.r.t. the ground truth. For each point of the evaluated model, we measure its distance from the ground truth.



(a) Ground truth (b) ReFusion (our approach) (c) StaticFusion

Figure 6.13: Models from our approach and StaticFusion of the scene *crowd3* compared against the ground truth. The points of the models are colored according to their distance from the ground truth. The arrow highlights the dynamic parts of the scene still present in the model from StaticFusion

For a qualitative impression, Figure 6.13 shows the two models of the scene *crowd3* where the points have been colored according to their distance to the closest point in the ground truth model. In Figure 6.13c, one can see that some dynamic elements are still present in the final model, represented by the red points highlighted by the arrow. In contrast, the model from our approach does not show such artifacts caused by dynamic objects.

For a quantitative evaluation, Figure 6.14 shows the cumulative percentage of points at a certain distance from the ground truth for the models of the two considered sequences. The plots show in both cases that the reconstructed model by our approach is more accurate.

In summary, our evaluation shows that our method is able to robustly track an RGB-D sensor in highly dynamic environments. At the same time, it provides a consistent and accurate model of the static part of the environment.

6.4 Related Work

Usually, mapping approaches assume a static environment and therefore handle moving objects mostly through outliers rejection. By discarding information that disagrees with the current measurements, one can handle implicitly dynamic objects [79]. Other approaches model the dynamic parts of the environment explicitly [62, 109] and filter these before the integration into the model. In this section, we cover works that explicitly deal with dynamic environments. Refer to Section 3.3 for an overview of SLAM techniques in static environments. We distinguish three main categories of SLAM approaches in dynamic environments: feature-based approaches, dense background reconstruction approaches, and dense approaches for non-rigidly deforming scenes.

The first category consists of approaches that are feature-based, but include some technique to distinguish features belonging to static objects, from features

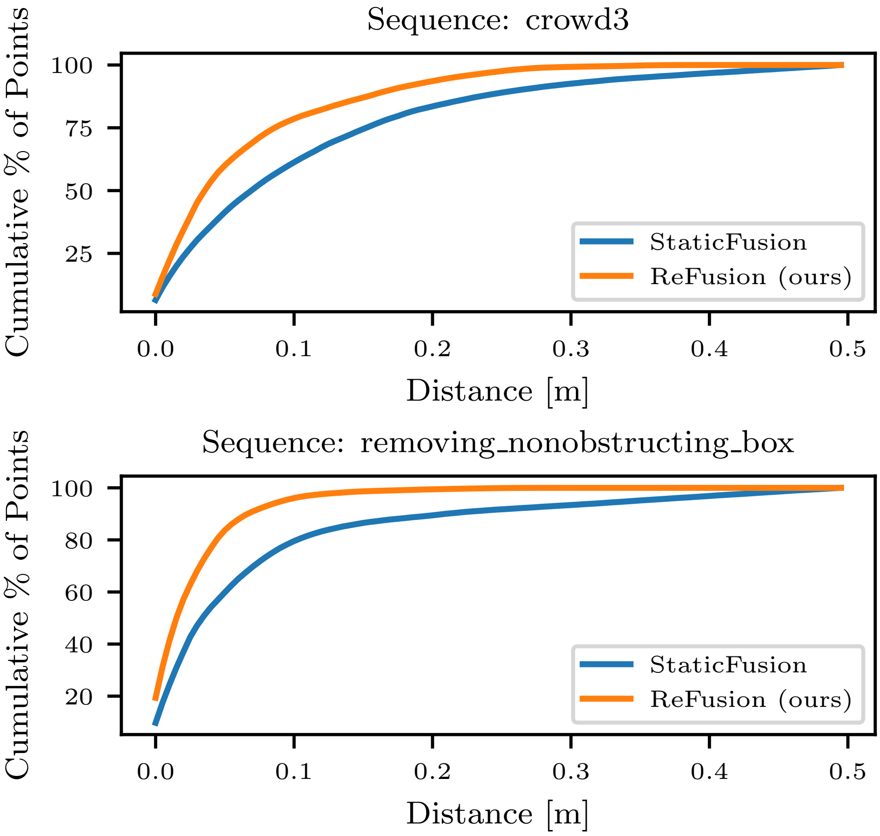


Figure 6.14: Plot of the cumulative percentage of points (y axis) at a specific distance from the ground truth (x axis).

detected on dynamic elements. A possible approach to this problem is to use an object tracker to track dynamic objects, and filter them out from the sensor data before performing SLAM. An example of this is the monocular SLAM pipeline proposed by Wangsiripitak *et al.* [138]. More recently, Riazuelo *et al.* [99] presented a standard feature-based visual SLAM algorithm, with the addition of an existing people tracker [57]. After estimating the pose of the sensor, they additionally create a dense 3D occupancy map of the environment, and provide, within this map, the full trajectories followed by the people in the environment. Another common approach is to compute the scene flow of the current frame w.r.t. the previous one and detect outliers by looking at the flow vectors. This is the approach followed for example by Alcantarilla *et al.* [2]. They propose an algorithm that uses a stereo camera and detect features using a Harris corner detector and MU-SURF descriptors. Using the computed features, they estimate the pose of the sensor using a pose graph-based SLAM algorithm, then compute the scene flow of the scene. Based on the flow, they remove the moving inliers and compute the pose of the sensor a second time. A similar approach is the one proposed by Wang *et al.* [137], who compute the scene flow of two consecutive RGB-D frames, and cluster features with similar 3D flow vectors. Then, they choose the largest cluster as the static part of the environment and use the features in a pose graph-based SLAM algorithm. Finally, a possible solution is to reproject features from a keyframe to the current frame to check whether they are static or dynamic. Tan *et al.* [129], for example, propose a monocular SLAM system that adopts this approach, in combination with a prior-based adaptive RANSAC. Bescos *et al.* [10] combine a similar geometric approach with a deep learning segmentation to enable removal of dynamics in an ORB-SLAM2 system [78]. Li *et al.* [68], in contrast to the previously described approaches, do not use standard features, but depth edge points. The points are weighted according to the distance between the real point and the one transformed from a keyframe, so that dynamic points influence less the SLAM algorithm.

The second category consists of approaches that focus on the dense reconstruction of the static part of the environment, discarding or modeling separately the dynamic parts. Keller *et al.* [62] use outliers in point correspondences during ICP as seed for segmentation of dynamics. Corresponding model surfels inside the segments are then marked as unstable. Kim *et al.* [64], instead, focus on extracting the background using a non-parametric model [35], from multiple depth images warped to be in the same pose. Using only the background information, they perform dense visual odometry. In contrast, Rünz *et al.* [100] explicitly track moving objects given by a segmentation process using either motion or semantic cues provided by class-agnostic object proposals [89]. In a more recent work [102], they use a deep learning segmentation, refined with a geometric approach, to detect objects in the environment. In addition, they reconstruct and track all the detected objects independently. A different approach is to cluster and track moving elements of the scene. For example, Sun *et al.* [126] first detect moving patches based on ego-motion-compensated image differencing. Then, they track the detected patches using a particle filter. Finally, they apply a maximum-a-posteriori estimator on depth images to determine the foreground. Similarly, Scona *et al.* [109] extend ElasticFusion [140] to incorporate only clusters that correspond to the static environment. Distinguishing static and dynamic parts of the environment is achieved by jointly estimating the camera pose and whether clusters are static or dynamic. This is computed by minimizing an energy function consisting of two terms: the first tries to align images by enforcing photometric and geometric consistency only for pixels that belong to static clusters, the second tries to distinguish static from dynamic clusters according to their residuals.

The third category includes the approaches that explicitly model dynamic elements and reconstruct them by allowing the environment to deform non-rigidly. The first approach to achieve this result in real-time is the one by Zollhöfer *et al.* [146]. They use a custom stereo camera to produce RGB-D data and they first need to acquire a template from scanning the object to reconstruct. Then they register in real-time the scans of the deforming object, by minimizing an energy function that encourage every visible model vertex to be as close as possible to the sample data, while using an as-rigid-as-possible regularizer, which controls the smoothness of deformations and motion. In a later work, Newcombe *et al.* [80] propose DynamicFusion, a template-free approach that runs in real-time on consumer RGB-D cameras. For each frame, they compute a warp field that transforms the model into the live frame, using an energy function similar to [146]. They estimate a 6D tranformation for sampled points on the mesh, then they interpolate them using dual-quaternion blending to obtain the full deformation field. Innmann *et al.* [54], with their VolumeDeform, extend DynamicFusion by adding sparse color feature matching, based on SIFT features, to compute the scene deformation. Guo *et al.* [47], instead, extend DynamicFusion by introducing albedo information to improve the reconstruction. Another extension is Fusion4D, by Dou *et al.* [31]. They introduce the possibility of combining the information from multiple RGB-D cameras. Moreover, instead of using just the first configuration of the environment as canonical pose, they use multiple key volumes that they warp to the current frame. In contrast to the previous approaches, Slavcheva *et al.* [113] do not represent the deformation field of points on a mesh, but they operate directly on the SDF. They minimize an energy function composed by a data term that aligns frames by minimizing the voxel-wise difference of signed distances, and a regularization term that enforces the deformation field to be approximately Killing, i.e., it generates locally nearly isometric motions. In a follow-up work, named SobolevFusion [114], they propose to estimate the flow field by defining the gradient flow in Sobolev space, leading to a faster and more detailed reconstruction. Moreover, they estimate the data association between voxels by matching the spectrum of the Laplacian matrix of shapes. Knowing the data association allows them to texture the model using RGB information. Finally, an approach that tries to combine static and dynamic reconstruction is MixedFusion, by Zhang *et al.* [143]. They employ a sigmoid-based ICP to detect dynamic objects. Then, they reconstruct the static part in a similar way as KinectFusion [79], and the dynamic parts in a similar way as DynamicFusion [80].

In this chapter, we propose an approach of the second category, i.e., we reconstruct the static part of the environment and discard dynamic elements. To detect dynamics, we rely on the residuals from the registration and on the detected free space. In this way, our approach is able to identify any kind of dynamics without relying on specific classes or models, and without explicitly tracking dynamic objects.

6.5 Conclusion

In this chapter, we presented ReFusion, a TSDF-based mapping approach able to track the pose of the camera in dynamic environments and build a consistent 3D model of the static world. Our approach tracks the sensor by exploiting directly the TSDF information and the color information encoded in voxel blocks that are only allocated when needed. Our method filters dynamics using an algorithm based on the residuals from the registration and the representation of free space. We evaluated our approach on the popular TUM RGB-D dataset, as well as on our Bonn RGB-D dynamic dataset, and provided comparisons to other state-of-the-art techniques. Our experiments show that our approach leads to an improved pose estimation and 3D reconstruction in the presence of dynamic elements in the environment, compared to other state-of-the-art dense SLAM approaches.

Chapter 7

Conclusion

Using a robot to build a 3D model of the environment has several attractive properties. An autonomous robotic platform exploring and mapping the environment eliminates the need of deploying trained personnel operating specialized equipment to scan the environment. This opens the possibility of using 3D models for several consumer applications, such as augmented reality or gaming, as well as for applications where the environment is not easily accessible by humans, such as rescue operations in disaster scenarios. In this thesis, we addressed several problems in the context of autonomous 3D reconstruction with robots. In particular, we presented an approach to fuse in real-time the measurements from the sensor into a dense 3D model, and we proposed a strategy to select the best viewpoints to take such measurements, in an unknown environment. Moreover, we tackled two different cases of non-static environments, by proposing an algorithm to detect long-term changes in a given 3D model from a sequence of images, and a technique to deal with dynamic elements, while we record the measurements. We tested all the approaches on real world data, and we designed our algorithms to work online on a robotic platform equipped with consumer-available sensors. In the remainder of this section, we briefly summarize the contributions of our work.

7.1 Summary of the Key Contributions

The first problem we addressed is the 3D reconstruction of the environment in real-time using an RGB-D sensor. We proposed a novel approach for dense SLAM able to track the sensor even in situation with scarce structural information. Moreover, our approach is implemented on the GPU and is efficient in terms of both execution time and memory consumption. This means that our approach is able to reconstruct larger areas compared to conventional grid-based methods, while maintaining a real-time performance, i.e., our approach is faster than the framerate of the sensor. We tested our algorithm on a publicly available dataset and provided a comparison with other state-of-the-art methods. Our experiments show that our approach performs better that the state of the art in situations with low structural cues.

The second problem we addressed is the selection of the best viewpoints to take the measurements necessary to build the 3D model of the environment autonomously. We proposed an approach that targets specifically MAVs, as they are particurlarly suitable for 3D exploration, due to their capabilities. Our algorithm selects iteratively the next best viewpoint that maximizes the information acquired from the sensor. Moreover, it takes into account the distance from the current position to the new point and the change of direction of motion. Finally, our method includes specific safety features in case the battery of the robot is about to come to an end. We tested our approach in both simulated and real environments, and compared it with other state-of-the-art methods. Our experiments show that our technique surpasses the state of the art in terms of uncertainty reduction, path smoothness and execution time.

The third problem we addressed is the detection of long-term changes in the environment, that have to be added in an outdated 3D model. We proposed a novel technique to detect changes between the current state of the environment and a given 3D model, using a short sequence of images. We detect the changes by reprojecting images onto each other, passing through the model. We then eliminate the ambiguities by combining the inconsistencies from multiple images. Finally, we triangulate the changes in 3D. The whole process takes less than two seconds, allowing it to be executed online on a robot. We tested our approach on existing datasets, as well as on our own, which we publicly share. Our experiments show that our method reliably identifies the changes in the environment w.r.t. an existing model, allowing a mapping or exploring robot to target those parts for its following measurements.

Finally, we targeted the problem of identifying dynamic elements in the measurements, with the purpose of filtering them out. We proposed an apprach that employs the residuals from the pose registration, in combination with an explicit representation of the free space, to build a consistent model of the static part of the environment. We tested our approach on a publicly available benchmark, as well as on our own dataset. The experiments show that our approach is more robust than the other tested dense SLAM algorithms, in terms of both tracking accuracy and consistency of the 3D model. In addition to our novel method, our contribution consists in a dataset for RGB-D SLAM containing several highly dynamic scenes. We provide for every scene the ground truth trajectory acquired with a motion capture system. In addition, we provide a ground truth 3D point cloud of the static environment recorded using a terrestrial laser scanner.

Overall, we proposed solutions to several important problems in the context of active 3D reconstruction with mobile robots. We demonstrated the effectiveness of our methods in real-world situations, often surpassing the state of the art. Although the four addressed challenges are not the only ones that arise when targeting the topic of this thesis, the algorithms that we proposed are crucial components of a fully working autonomous exploring robot, and can be directly integrated in the complete pipeline of the system.

7.2 Open Source Contributions

The work described in this thesis led to the release of datasets and open source software. Here we provide a list of links where the software and the datasets are published:

- Fast Change Detection C++ Library, presented in Chapter 5:
  `https://github.com/PRBonn/fast_change_detection`
- Change Detection Dataset, presented in Chapter 5:
  `http://www.ipb.uni-bonn.de/data/changedetection2017`
- ReFusion C++ Library, presented in Chapter 6:
  `https://github.com/PRBonn/refusion`
- Bonn RGB-D Dynamic Dataset, presented in Chapter 6:
  `http://www.ipb.uni-bonn.de/data/rgbd-dynamic-dataset`

Bibliography

[1] P.F. Alcantarilla, S. Stent, G. Ros, R. Arroyo, and R. Gherardi. Street-View Change Detection with Deconvolutional Networks. In *Proc. of Robotics: Science and Systems (RSS)*, 2016.

[2] P.F. Alcantarilla, J.J. Yebes, J. Almazán, and L.M. Bergasa. On Combining Visual SLAM and Dense Scene Flow to Increase the Robustness of Localization and Mapping in Dynamic Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1290–1297, 2012.

[3] J. Aloimonos, I. Weiss, and A. Bandyopadhyay. Active Vision. *Intl. Journal of Computer Vision (IJCV)*, 1(4):333–356, 1988.

[4] R. Ambruş, N. Bore, J. Folkesson, and P. Jensfelt. Meta-Rooms: Building and Maintaining Long Term Spatial Models in a Dynamic World. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1854–1861, 2014.

[5] H. Andreasson, M. Magnusson, and A. Lilienthal. Has Somethong Changed Here? Autonomous Difference Detection for Security Patrol Robots. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3429–3435, 2007.

[6] H. Andreasson, R. Triebel, and W. Burgard. Improving Plane Extraction from 3D Data by Fusing Laser Data and Vision. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2656–2661, 2005.

[7] N. Atanasov, J. Le Ny, K. Daniilidis, and G.J. Pappas. Decentralized Active Information Acquisition: Theory and Application to Multi-Robot SLAM. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4775–4782, 2015.

[8] S. Bai, F. Chen, and B. Englot. Toward Autonomous Mapping and Exploration for Mobile Robots through Deep Supervised Learning. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[9] R. Bajcsy. Active Perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[10] B. Bescos, J.M. Fácil, J. Civera, and J. Neira. DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):4076–4083, 2018.

[11] P.J. Besl and N.D. McKay. A Method for Registration of 3D Shapes. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 14(2):239–256, 1992.

[12] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart. Structural Inspection Path Planning via Iterative Viewpoint Resampling with Application to Aerial Robotics. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 6423–6430, 2015.

[13] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding Horizon ”Next-Best-View” Planner for 3D Exploration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[14] F. Bourgault, A.A. Makarenko, S.B. Williams, B. Grocholsky, and H.F. Durrant-Whyte. Information Based Adaptive Robotic Exploration. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 1, pages 540–545, 2002.

[15] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions. In *Proc. of Robotics: Science and Systems (RSS)*, volume 2, 2013.

[16] D. Canelhas, T. Stoyanov, and A. Lilienthal. SDF Tracker: A Parallel Algorithm for On-Line Pose Estimation and Scene Reconstruction from Depth Images. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3671–3676, 2013.

[17] T. Caselitz, B. Steder, M. Ruhnke, and W. Burgard. Monocular Camera Localization in 3D LiDAR Maps. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[18] 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 *Proc. of Robotics: Science and Systems (RSS)*, 2015.

[19] B. Chen, Z. Chen, L. Deng, Y. Duan, and J. Zhou. Building Change Detection with RGB-D Map Generated from UAV Images. *Neurocomputing*, 208:350–364, 2016.

[20] J. Chen, D. Bautembach, and S. Izadi. Scalable Real-Time Volumetric Surface Reconstruction. *ACM Transactions on Graphics*, 32(4):113, 2013.

[21] K. Choi, I. Lee, and S. Kim. A Feature Based Approach to Automatic Change Detection from LiDAR Data in Urban Areas. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 18:259–264, 2009.

[22] S. Choi, Q. Zhou, and V. Koltun. Robust Reconstruction of Indoor Scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 5556–5565, 2015.

[23] T. Cieslewski, E. Kaufmann, and D. Scaramuzza. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High Speed Flight. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[24] D. Crispell, J. Mundy, and G. Taubin. A Variable-Resolution Probabilistic Three-Dimensional Model for Change Detection. *IEEE Trans. on Geoscience and Remote Sensing*, 50(2):489–500, 2012.

[25] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 303–312. ACM, 1996.

[26] A. Dai, A.X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[27] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. BundleFusion: Real-Time Globally Consistent 3D Reconstruction using On-the-fly Surface Reintegration. *ACM Transactions on Graphics*, 36(4):76a, 2017.

[28] B. Della Corte, I. Bogoslavskyi, C. Stachniss, and G. Grisetti. A General Framework for Flexible Multi-Cue Photometric Point Cloud Registration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[29] J. Delmerico, S. Isler, R. Sabzevari, and D. Scaramuzza. A Comparison of Volumetric Information Gain Metrics for Active 3D Object Reconstruction. *Autonomous Robots*, pages 1–12, 2017.

[30] G.R. Dini, K. Jacobsen, F. Rottensteiner, M. Al Rajhi, and C. Heipke. 3D Building Change Detection using High Resolution Stereo Images and a GIS Database. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 39:299–304, 2012.

[31] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S.R. Fanello, A. Kowdle, S.O. Escolano, C. Rhemann, D. Kim, J. Taylor, K. Pushmeet, V. Tankovich, and S. Izadi. Fusion4D: Real-time Performance Capture of Challenging Scenes. *ACM Trans. on Graphics (TOG)*, 35(4):114, 2016.

[32] A.D. Dragan, K.C.T. Lee, and S.S. Srinivasa. Legibility and Predictability of Robot Motion. In *Proc. of the ACM/IEEE Intl. Conf. on Human-Robot Interaction*, 2013.

[33] E. Dunn and J. Frahm. Next Best View Planning for Active Model Improvement. In *Proc. of British Machine Vision Conference (BMVC)*, pages 1–11, 2009.

[34] I. Eden and D.B. Cooper. Using 3D Line Segments for Robust and Efficient Change Detection from Multiple Noisy Images. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 172–185, 2008.

[35] A. Elgammal, D. Harwood, and L. Davis. Non-parametric Model for Background Subtraction. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 751–767, 2000.

[36] M. Everingham, L. Van Gool, C.K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *Intl. Journal of Computer Vision (IJCV)*, 88(2):303–338, 2010.

[37] M. Fehr, M.T. Dymczyk, S. Lynen, and R. Siegwart. Reshaping Our Model of the World Over Time. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[38] M. Fehr, F. Furrer, I. Dryanovski, J. Sturm, I. Gilitschenski, R. Siegwart, and C.C. Lerma. TSDF-Based Change Detection for Consistent Long-Term Dense Reconstruction and Dynamic Object Discovery. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[39] R. Finman, T. Whelan, M. Kaess, and J.J. Leonard. Toward Lifelong Object Segmentation from Change Detection in Dense RGB-D Maps. In *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, pages 178–185, 2013.

[40] A.W. Fitzgibbon. Robust Registration of 2D and 3D Point Sets. *Journal on Image and Vision Computing (IVC)*, 21(13-14):1145–1153, 2003.

[41] C. Forster, M. Pizzoli, and D. Scaramuzza. Appearance-based Active, Monocular, Dense Reconstruction for Micro Aerial Vehicles. In *Proc. of Robotics: Science and Systems (RSS)*, 2014.

[42] W. Förstner and B. Wrobel. *Photogrammetric Computer Vision – Statistics, Geometry, Orientation and Reconstruction.* Springer Verlag, 2016.

[43] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. Vision-Based Autonomous Mapping and Exploration Using a Quadrotor MAV. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4557–4564, 2012.

[44] Charles Freundlich, Philippos Mordohai, and Michael M Zavlanos. A Hybrid Control Approach to the Next-Best-View Problem using Stereo Vision. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4493–4498, 2013.

[45] D. Girardeau-Montaut, M. Roux, R. Marc, and G. Thibault. Change Detection on Points Cloud Data Acquired with a Ground Laser Scanner. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(part 3):W19, 2005.

[46] M. Golparvar-Fard, F. Pena-Mora, and S. Savarese. Monitoring Changes of 3D Building Elements from Unordered Photo Collections. In *Proc. of the Int. Conf. on Computer Vision (ICCV) Workshops*, pages 249–256, 2011.

[47] K. Guo, F. Xu, T. Yu, X. Liu, Q. Dai, and Y. Liu. Real-Time Geometry, Albedo, and Motion Reconstruction using a Single RGB-D Camera. *ACM Trans. on Graphics (TOG)*, 36(3):32, 2017.

[48] Christof H., Manfred K., Markus R., Andreas W., Stefan K., Horst B., and Gerhard R. Online Feedback for Structure-from-Motion Image Acquisition. In *Proc. of British Machine Vision Conference (BMVC)*, pages 70.1–70.12, 2012.

[49] S. Haner and A. Heyden. Optimal View Path Planning for Visual SLAM. In *Proc. of the Scandinavian Conference on Image Analysis*, pages 370–380, 2011.

[50] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[51] A.J. Heller, Y.G. Leclerc, and Q. Luong. Framework for Robust 3D Change Detection. In *Sensors, Systems, and Next-Generation Satellites*, volume 4540, pages 639–650, 2001.

[52] A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 34:189–206, 2013.

[53] M. Hussain, D. Chen, A. Cheng, H. Wei, and D. Stanley. Change Detection from Remotely Sensed Images: From Pixel-Based to Object-Based Approaches. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 80:91–106, 2013.

[54] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger. VolumeDeform: Real-time Volumetric Non-rigid Reconstruction. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 362–379, 2016.

[55] S.R. Isler, R. Sabzevari, J. Delmerico, and D. Scaramuzza. An Information Gain Formulation for Active Volumetric 3D Reconstruction. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[56] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction using a Moving Depth Camera. In *ACM Symposium on User Interface Software and Technology*, pages 559–568, 2011.

[57] O.H. Jafari, D. Mitzel, and B. Leibe. Real-Time RGB-D Based People Detection and Tracking for Mobile Robots and Head-Worn Cameras. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 5636–5643, 2014.

[58] S.J. Julier and J.K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. *Proc. of the SPIE Conf. on Reconnaissance and Electronic Warfare System*, 3068:182–193, 1997.

[59] O. Kähler, 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 Trans. on Visualization and Computer Graphics*, 21(11):1241–1250, 2015.

[60] O. Kähler, V.A. Prisacariu, J. Valentin, and D. Murray. Hierarchical Voxel Block Hashing for Efficient Integration of Depth Images. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[61] Z. Kang and Z. Lu. The Change Detection of Building Models Using Epochs of Terrestrial Point Clouds. In *Proc. of the IEEE Intl. Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping*, pages 1–6, 2011.

[62] M. Keller, D. Lefloch, M. Lambers, and S. Izadi. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, pages 1–8, 2013.

[63] C. Kerl, J. Sturm, and D. Cremers. Robust Odometry Estimation for RGB-D Cameras. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3748–3754, 2013.

[64] D.H. Kim and J.H. Kim. Effective Background Model-Based RGB-D Dense Visual Odometry in a Dynamic Environment. *IEEE Trans. on Robotics (TRO)*, 32(6):1565–1573, 2016.

[65] J. Košecka. Detecting Changes in Images of Street Scenes. In *Proc. of the Asian Conf. on Computer Vision (ACCV)*, pages 590–601, 2012.

[66] M. Krainin, B. Curless, and D. Fox. Autonomous Generation of Complete 3D Object Models Using Next Best View Manipulation Planning. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 5031–5037, 2011.

[67] S. Kriegel, T. Bodenmüller, M. Suppa, and G. Hirzinger. A Surface-Based Next-Best-View Approach for Automated 3D Model Completion of Unknown Objects. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4869–4874, 2011.

[68] S. Li and D. Lee. RGB-D SLAM in Dynamic Environments using Static Point Weighting. *IEEE Robotics and Automation Letters (RA-L)*, 2(4):2263–2270, 2017.

[69] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 740–755, 2014.

[70] W.E. Lorensen and H.E. Cline. Marching Cubes: a High Resolution 3D Surface Construction Algorithm. In *Proc. of the Intl. Conf. on Computer*

Graphics and Interactive Techniques (SIGGRAPH), volume 21, pages 163–169, 1987.

[71] D. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2004.

[72] J.A. Malpica, M.C. Alonso, F. Papí, A. Arozarena, and A. Martínez De Agirre. Change Detection of Buildings from Satellite Imagery and LiDAR Data. *International Journal of Remote Sensing*, 34(5):1652–1675, 2013.

[73] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger. Fusion++: Volumetric Object-Level SLAM. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, pages 32–41, 2018.

[74] D. Meagher. Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer. *Technical Report*, 1980.

[75] A. Milioto and C. Stachniss. Bonnet: An Open-Source Training and Deployment Framework for Semantic Segmentation in Robotics using CNNs. *Worshop on Perception, Inference, and Learning for Joint Semantic, Geometric, and Physical Understanding, IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2018.

[76] 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 *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[77] C. Mostegel, A. Wendel, and H. Bischof. Active Monocular Localization: Towards Autonomous Monocular Exploration for Multirotor MAVs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3848–3855, 2014.

[78] R. Mur-Artal and J.D. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Trans. on Robotics (TRO)*, 2017.

[79] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proc. of the Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[80] R.A. Newcombe, D. Fox, and S.M. Seitz. DynamicFusion: Reconstruction and Tracking of Non-Rigid Scenes in Real-Time. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, 2015.

[81] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. *Proc. of the SIGGRAPH Asia*, 32(6), 2013.

[82] M. Nieuwenhuisen and S. Behnke. Layered Mission and Path Planning for MAV Navigation with Partial Environment Knowledge. In *Intelligent Autonomous Systems 13*, pages 307–319. Springer, 2016.

[83] P. Núñez, P. Drews, A. Bandera, R. Rocha, M. Campos, and J. Dias. Change Detection in 3D Environments Based on Gaussian Mixture Model and Robust Structural Matching for Autonomous Robotic Applications. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2633–2638, 2010.

[84] E. Palazzolo, J. Behley, P. Lottes, P. Giguère, and C. Stachniss. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[85] E. Palazzolo and C. Stachniss. Change Detection in 3D Models Based on Camera Images. In *9th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[86] E. Palazzolo and C. Stachniss. Information-Driven Autonomous Exploration for a Vision-Based MAV. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W3:59–66, 2017.

[87] E. Palazzolo and C. Stachniss. Effective Exploration for MAVs Based on the Expected Information Gain. *Drones*, 2(1), 2018.

[88] E. Palazzolo and C. Stachniss. Fast Image-Based Geometric Change Detection Given a 3D Model. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[89] P.O. Pinheiro, T. Lin, R. Collobert, and P. Dollár. Learning to Refine Object Segments. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 75–91, 2016.

[90] R. Pito. A Solution to the Next Best View Problem for Automated Surface Acquisition. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 21(10):1016–1030, 1999.

[91] M. Pizzoli, C. Forster, and D. Scaramuzza. REMODE: Probabilistic, Monocular Dense Reconstruction in Real Time. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2609–2616, 2014.

[92] T. Pollard and J.L. Mundy. Change Detection in a 3D World. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–6, 2007.

[93] C. Potthast and G.S. Sukhatme. A Probabilistic Framework for Next Best View Estimation in a Cluttered Environment. *Journal of Visual Communication and Image Representation (JVCIR)*, 25(1):148–164, 2014.

[94] R. Qin and A. Gruen. 3D Change Detection at Street Level Using Mobile Laser Scanning Point Clouds and Terrestrial Images. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 90:23–35, 2014.

[95] R. Qin, J. Tian, and P. Reinartz. 3D Change Detection – Approaches and Applications. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 122:41–56, 2016.

[96] P. Quin, G. Paul, A. Alempijevic, D. Liu, and G. Dissanayake. Efficient Neighbourhood-Based Information Gain Approach for Exploration of Complex 3D Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1343–1348, 2013.

[97] R.J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image Change Detection Algorithms: a Systematic Survey. *IEEE Trans. on Image Processing*, 14(3):294–307, 2005.

[98] M. Reinhardt, B. Noack, and U.D. Hanebeck. Closed-Form Optimization of Covariance Intersection for Low-Dimensional Matrices. In *Proc. of the Int. Conf. on Information Fusion*, pages 1891–1896, 2012.

[99] L. Riazuelo, L. Montano, and J.M.M. Montiel. Semantic Visual SLAM in Populated Environments. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–7, 2017.

[100] M. Runz and L. Agapito. Co-Fusion: Real-Time Segmentation, Tracking and Fusion of Multiple Objects. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[101] M. Rünz and L. Agapito. MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects. *arXiv preprint*, 2018.

[102] M. Rünz, M. Buffier, and L. Agapito. MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects. In *Proc. of the Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 10–20, 2018.

[103] S.A. Sadat, K. Chutskoff, D. Jungic, J. Wawerla, and R. Vaughan. Feature-Rich Path Planning for Robust Navigation of MAVs with Mono-SLAM. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3870–3875, 2014.

[104] K. Sakurada and T. Okatani. Change Detection from a Street Image Pair using CNN Features and Superpixel Segmentation. In *Proc. of British Machine Vision Conference (BMVC)*, pages 61–1, 2015.

[105] K. Sakurada, T. Okatani, and K. Deguchi. Detecting Changes in 3D Structure of a Scene from Multi-View Images Captured by a Vehicle-Mounted Camera. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 137–144, 2013.

[106] H. Sarbolandi, D. Lefloch, and A. Kolb. Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect. *Journal of Computer Vision and Image Understanding (CVIU)*, 139:1–20, 2015.

[107] K. Schmid, H. Hirschmüller, A. Dömel, I. Grixa, M. Suppa, and G. Hirzinger. View Planning for Multi-View Stereo 3D Reconstruction using an Autonomous Multicopter. *Journal of Intelligent and Robotic Systems (JIRS)*, 65(1-4):309–323, 2012.

[108] J. Schneider, C. Eling, L. Klingbeil, H. Kuhlmann, W. Förstner, and C. Stachniss. Fast and Effective Online Pose Estimation and Mapping for UAVs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 4784–4791, 2016.

[109] R. Scona, M. Jaimez, Y.R. Petillot, M. Fallon, and D. Cremers. StaticFusion: Background Reconstruction for Dense RGB-D SLAM in Dynamic Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[110] W. Scott, G. Roth, and J. Rivest. View Planning for Automated 3D Object Reconstruction Inspection. *ACM Computing Surveys*, 35(1), 2003.

[111] R. Shade and P. Newman. Choosing Where to Go: Complete 3D Exploration with Stereo. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2806–2811, 2011.

[112] S. Shen, N. Michael, and V. Kumar. Autonomous Indoor 3D Exploration with a Micro-Aerial Vehicle. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 9–15, 2012.

[113] M. Slavcheva, M. Baust, D. Cremers, and S. Ilic. KillingFusion: Non-Rigid 3D Reconstruction without Correspondences. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1386–1395, 2017.

[114] M. Slavcheva, M. Baust, and S. Ilic. SobolevFusion: 3D Reconstruction of Scenes Undergoing Free Non-Rigid Motion. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2646–2655, 2018.

[115] M. Slavcheva and S. Ilic. SDF-TAR: Parallel Tracking and Refinement in RGB-D Data using Volumetric Registration. In *Proc. of British Machine Vision Conference (BMVC)*, 2016.

[116] M. Slavcheva, W. Kehl, N. Navab, and S. Ilic. SDF-2-SDF: Highly Accurate 3D Object Reconstruction. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 680–696, 2016.

[117] M. Slavcheva, W. Kehl, N. Navab, and S. Ilic. SDF-2-SDF Registration for Real-Time 3D Reconstruction from RGB-D Data. *Intl. Journal of Computer Vision (IJCV)*, pages 1–22, 2018.

[118] C. Stachniss, G. Grisetti, and W. Burgard. Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. In *Proc. of Robotics: Science and Systems (RSS)*, pages 65–72, Cambridge, MA, USA, 2005.

[119] F. Steinbrucker, C. Kerl, and D. Cremers. Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, pages 3264–3271, 2013.

[120] F. Steinbrücker, J. Sturm, and D. Cremers. Real-Time Visual Odometry from Dense RGB-D Images. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, pages 719–722, 2011.

[121] F. Steinbrücker, J. Sturm, and D. Cremers. Volumetric 3D Mapping in Real-Time on a CPU. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[122] S. Stent, R. Gherardi, B. Stenger, and R. Cipolla. Detecting Change for Multi-View, Long-Term Surface Inspection. In *Proc. of British Machine Vision Conference (BMVC)*, pages 127–1, 2015.

[123] H. Strasdat, C. Stachniss, and W. Burgard. Which Landmark is Useful? Learning Selection Policies for Navigation in Unknown Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, Kobe, Japan, 2009.

[124] D. Perea Ström, F. Nenci, and C. Stachniss. Predictive Exploration Considering Previously Mapped Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2015.

[125] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[126] Y. Sun, M. Liu, and M.Q.H. Meng. Improving RGB-D SLAM in Dynamic Environments: a Motion Removal Approach. *Journal on Robotics and Autonomous Systems (RAS)*, 89:110–122, 2017.

[127] S. Suzuki and K. Abe. Topological Structural Analysis of Digitized Binary Images by Border Following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[128] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper With Convolutions. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[129] W. Tan, H. Liu, Z. Dong, G. Zhang, and H. Bao. Robust Monocular SLAM in Dynamic Environments. In *Proc. of the Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 209–218, 2013.

[130] A. Taneja, L. Ballan, and M. Pollefeys. Image Based Detection of Geometric Changes in Urban Environments. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, pages 2336–2343, 2011.

[131] A. Taneja, L. Ballan, and M. Pollefeys. City-Scale Change Detection in Cadastral 3D Models Using Images. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 113–120, 2013.

[132] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M.H. Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects.

In *Proc. of Vision, Modeling, Visualization (VMV)*, volume 3, pages 47–54, 2003.

[133] M. Trummer, C. Munkelt, and J. Denzler. Online Next-Best-View Planning for Accuracy Optimization Using an Extended E-Criterion. In *Proc. of the Intl. Conf. on Pattern Recognition (ICPR)*, pages 1642–1645, 2010.

[134] A.O. Ulusoy and J.L. Mundy. Image-Based 4D Reconstruction Using 3D Change Detection. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 31–45, 2014.

[135] J.I. Vasquez-Gomez, L.E. Sucar, R. Murrieta-Cid, and E. Lopez-Damian. Volumetric Next-Best-View Planning for 3D Object Reconstruction with Positioning Error. *Intl. Journal of Advanced Robotic Systems*, 11, 2014.

[136] A.S. Vempati, I. Gilitschenski, J. Nieto, P. Beardsley, and R. Siegwart. Onboard Real-time Dense Reconstruction of Large-scale Environments for UAV. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[137] Y. Wang and S. Huang. Motion Segmentation Based Robust RGB-D SLAM. In *Proc. of the World Congress on Intelligent Control and Automation*, pages 3122–3127, 2014.

[138] S. Wangsiripitak and D.W. Murray. Avoiding Moving Outliers in Visual SLAM by Tracking Moving Objects. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 375–380, 2009.

[139] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially Extended KinectFusion. In *Proc. RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[140] T. Whelan, S. Leutenegger, R. S. Moreno, B. Glocker, and A. Davison. ElasticFusion: Dense SLAM Without A Pose Graph. In *Proc. of Robotics: Science and Systems (RSS)*, 2015.

[141] W. Xiao, B. Vallet, M. Brédif, and N. Paparoditis. Street Environment Change Detection from Mobile Laser Scanning Point Clouds. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 107:38–49, 2015.

[142] B. Yamauchi. A Frontier-Based Approach for Autonomous Exploration. In *Proc. of the IEEE Intl. Symp. on Computer Intelligence in Robotics and Automation (CIRA)*, pages 146–151, 1997.

[143] H. Zhang and F. Xu. MixedFusion: Real-Time Reconstruction of an Indoor Scene with Dynamic Objects. *IEEE Trans. on Visualization and Computer Graphics*, 24(12):3137–3146, 2018.

[144] Q. Zhou and V. Koltun. Dense Scene Reconstruction with Points of Interest. *ACM Transactions on Graphics*, 32(4):112, 2013.

[145] Q. Zhou, S. Miller, and V. Koltun. Elastic Fragments for Dense Scene Reconstruction. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, pages 473–480, 2013.

[146] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt, and M. Stamminger. Real-Time Non-Rigid Reconstruction using an RGB-D Camera. *ACM Trans. on Graphics (TOG)*, 33(4):156, 2014.

[147] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. State of the Art on 3D Reconstruction with RGB-D Cameras. In *Eurographics - State-of-the-Art Reports (STARs)*, volume 37, 2018.

List of Figures

List of Tables

List of Algorithms