Technische Universität München
Fakultät für Informatik

Technische Universiteit Eindhoven
Faculteit Electrical Engineering

Doctoral Thesis

Semantic Mapping and Tracking with RGB-D Cameras

Lingni Ma

Supervised By

Prof. Dr. Daniel Cremers
Prof. Dr. Peter H. N. de With

July 2018

Technical University of Munich

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik

Lehrstuhl für Bildverarbeitung und Mustererkennung

Semantic Mapping and Tracking with RGB-D Cameras

Lingni Ma

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Nassir Navab (Technische Universität München)
Prüfer der Dissertation:	1. Prof. Dr. Daniel Cremers (Technische Universität München)
	2. Prof. Dr. Peter H. N. de With (Technische Universiteit Eindhoven)
	3. Prof. Dr. Matthias Niessner (Technische Universität München)
	4. Prof. Dr. Jack van Wijk (Technische Universiteit Eindhoven)
	5. Prof. Dr. Richard A. Newcombe (University of Washington)
	6. Prof. Dr. Thomas Brox (Albert-Ludwigs-Universität Freiburg)

Die Dissertation wurde am 10.08.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.08.2019 angenommen.

TECHNISCHE UNIVERSITEIT EINDHOVEN

Semantic Mapping and Tracking with RGB-D Cameras

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor Promoties,
in het openbaar te verdedigen
op donderdag 14 januari 2021 om 12.00 uur

door

Lingni Ma

geboren te Sichuan, China

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommisie is als volgt:

voorzitter:	Prof. Dr. Nassir Navab (Technische Universität München), nassir.navab@tum.de
1e promotor:	Prof. Dr. Daniel Cremers (Technische Universität München)
2e promotor:	Prof. Dr. Peter H. N. de With
leden:	Prof. Dr. Matthias Niessner (Technische Universität München)
	Prof. Dr. Jack van Wijk
	Prof. Dr. Richard A. Newcombe (University of Washington)
	Prof. Dr. Thomas Brox (Albert-Ludwigs-Universität Freiburg)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

To my beloved mom and dad.

All I ever want is to create something beautiful.

Abstract

DENSE visual mapping and tracking is a fundamental problem in computer vision and robotics. Many existing algorithms that explore the direct image methods and the low-level features, while less investigated is how to infer and aggregate high-level understandings. This thesis aims to bridge this gap for RGB-D vision from several perspectives, including dense 3D reconstruction, visual simultaneous localization and mapping (SLAM) and machine learning. Our focuses hence are i) algorithms that extract high-level abstracts and semantics from RGB-D data; and ii) methods that utilize such information to optimize dense mapping, improve motion estimation and obtain meaningful scene representations.

One major contribution is using plane priors for mapping and tracking. Planar surfaces are common features of indoor scenes, which often encodes semantics, *e.g.,* walls, floors and windows. This thesis developed methods to detect planar surfaces from different data source. Using plane priors, we introduced algorithms to reduce redundancy in dense indoor scanning and obtain compact reconstruction without losing geometrical and visual information. Further, we proposed a full real-time RGB-D SLAM algorithm, known as CPA SLAM, which models the indoor scenes with global planar maps, estimates camera motion with a mixture of keyframe-based and model-based tracking method and optimizes the global consistency using the planar maps.

Beyond plane priors, we contributed several algorithms to extract semantics from RGB-D vision using deep convolutional neural networks (CNNs). Firstly we introduced FuseNet, a novel network architecture that is shown to be efficient in learning features from single-view RGB-D images. Further we proposed algorithms to regularize CNNs with multi-view constraints based on geometry from RGB-D sequences and showed the better consistent semantic mapping can be obtained. Another contribution is the introduction of discrete wavelet transform (DWT) into encoder-decoder CNNs. Using DWT and inverse DWT to perform unpooling and develop pyramids to obtain global context, we improved the level of details in dense semantic segmentation.

Our last contribution is a semantic annotation algorithm with a close-loop labeling scheme. A segmentation-based free-form mesh labeling with the human-aided scene completion is proposed to produce consistent 3D labeled meshes and 2D annotated videos. Additionally, we developed algorithms to obtain instance segmentation for 3D meshes by learning and fusing 2D segmentation from annotated data. This close-loop labeling scheme is shown beneficial for large-scale ground truth generation.

Acknowledgments

DURING my pursuit of PhD, I am very fortunate to have the chance to learn from and work with many gifted experts. I am also very lucky to have met a lot of wonderful people, who have supported me and brought joy along the journey. Here, I would like to express my heartfelt gratitude.

My early PhD was spent at TU Eindhoven in the Netherlands. I thank prof. Peter H. N. de With, who introduced me to 3D computer vision and gave me the chance to start PhD in his group. As a busy professor, Peter spent many of his precious evenings teaching me how to write properly. I sincerely thank Peter for very kindly supporting my wish for joint PhD with TU Munich and for his effort in making the dual promotion at all possible. I thank all the nice people I have closely worked with, Egor Bondarev, Luat Do, Raphael Favier, Francisco Heredia, Frits Florentinus (I hope you have won the battle now). I thank our secretary Anja de Valk for helping me organizing all the paperwork, and warm my heart with her love and care.

I spent most of my PhD in TU Munich. I thank prof. Daniel Cremers, for accepting me as a transfer student and gave me the chance to grow stronger and dive deeper in research. Daniel introduced me the power of variational methods and optimization. I thank him for opening many possibilities in my career and building an excellent group. I thank Jörg Stückler, Christian Kerl, and Jakob Engel, who kindly shared their knowledge about SLAM and inspired me to persue my ideas. My colleagues and friends, Oriana Baldizan, Julia Diebold, John Chiotellis, Mariano Jaimez Tarifa, Vladyslav Usenko, Robert Maier, Mohamed Souiai, Philip Häusser, Thomas Möllenhoff, Christian Sommer, David Schubert, Yvain Queau, Vladimir Golkov, Caner Hazirbas and Csaba Domokos, I thank them for all the delightful conversions. My great gratitude for our secretary Sabine Wagner and system administrator Quirin Lohr, for their kind effort in organizing paperwork and always reliable IT support.

My last year in Seattle was a surreal dream, to work among some best professionals and be part of the amazing research. Richard Newcombe, thank you for offering me the two fantastic internships with Surreal at Oculus Research and for building the truly amazing research team. The discussions I had with Richard were one of the most refreshing and inspiring ones. Thomas Whelan, I thank him for being the coolest intern manager anyone could wish for, who is supportive, encouraging and inspiring. It is also my luck to know Tom at an early stage of my PhD. He has always been my role model, who shown me how to do science and program properly. Julian Straub, thank you for claiming the title of my best friend and acting as one. Steven Chen, thank you for all the critical suggestions to make Nibbler better. Jakob, thank you for all the helps with Highgate, which made everything else possible. Yuyang Zou, thank you for patiently testing and providing valuable feedbacks to improve Nibbler. Also thank you for being my sunshine who drives away the shades when shit happens and to put back my smile. Svetoslav Kolev, Rohan Chabra, Alberto García García, thank you for keeping me accompany with valuable discussions many late working nights and weekends. Paul Yon, thank you for patiently teaching me the latte art and for bringing Nibbler into an awesome product. Renzo De Nardi, thank you for always having an answer to my whatever question. People I have worked with and learned from, Michael Goesele, David Rosen, Steven Lovegrove, I thank them for sharing their knowledge and patience in explaining.

I thank all the committee members, Daniel, Peter, Richard, Michael, prof. Matthias Nießner and prof. Jack van Wijk, who kindly review this thesis and provide their professional feedbacks given a short amount of time. Fons van der Sommen and Willem P. Sanberg thank you for kindly translate the thesis abstract into Dutch, and Zorah Lähner, thank you for the German translation. Although I do not understand them, they are beautiful. My gratitude to Robert and John for the wonderful proof reading and suggestions. I am also thank Manuela Fischer for her kind effort in helping me organizing the joint doctoral.

All my dearest friends, I thank them for filling up my life with joy and excitement. Unique van de Biezenbos and Alina Fan, you are my family. The years we spent together in school, the fun trips around Europe, your always-ready life-saving suggestions, amazing boardgames and pooling lessons, will always be my sweetest memories about the Netherlands. Zorah, thank you for being my missing little sister. I do not know where to start to list all the sweet and fun things we experienced together. Thank you for all the crazy laughs, ice-cream and wine to celebrate each good time and all the hugs and encouragement you offered to get me through the difficulties. Alice Ya, thank you for an amazing friendship since high school. People who appeared in my life and made me a better person, thank you for keeping me nerdy, bringing me peace, and being my inspiration.

Last but the most important, my beloved mom and dad. Despite years apart, I would not be where I am, having what I have, without you. You patiently taught me all the most important lessons about life and planted the curiosity in me to explore the world. Throughout the years, you have always encouraged me to live my dreams and fight for my believes with all my heart. The home you provide me with, is the warmest shelter I can always escape to find peace, strength and joy no matter what. Thank you for nurturing me into a happy, brave and self-sufficient person and thank you being truly proud of me. I am forever in debt to your unconditional love and endless support.

The journey has been long. Luckily with all my wonderful colleagues, friends and family, it was a fun one. Thank you all.

Contents

PART I

Introduction and Preliminaries

Nobody ever figures out what life is all about, and it doesn't matter. Explore the world. Nearly everything is really interesting if you go into it deeply enough.

– Richard P. Feynman

CHAPTER 1

Introduction

APPROXIMATELY half a century ago, the problem - *"let computer describe what it sees"* started to intrigue computer scientists, machine learning experts, and mathematicians. Throughout the years of efforts, researchers have been pushing the limits of machine vision to approximate the human visual system. However, many tasks that are natural and easy for human reception, such as parsing visual contents up to any level of details, identifying foreign stuffs as homogeneous objects, relating alien matters to familiar concepts, recognizing correspondences under extreme changes, still present significant challenges for machine vision.

Within the big question and towards the application end where machine perception and understandings are fundamental, consider the scenarios where an autonomous device need to navigate around the environment and interact with its surroundings. Three problems are confronted: how to map the environment, how to localize the device onto the map, and what decisions to make. With decades of research, people develop solutions by modeling the physical mechanics of the scene. More recent research start to explore the direction that integrates abstract-level machine reasoning and exploits semantic visual understandings to assist mapping, tracking and ultimately decision making. This direction where artificial intelligence is employed to tackle the visual navigation and reconstruction is a closer approximation to how humans navigate and interact with the world.

1.1 Motivation and Scope

Naturally for humans, we use semantic concepts and understandings in almost every aspect of our behavior. When we navigate and interact with the physical world, our perception and reasoning is rather abstract than primitive. For humans, we often think of the world as a composition of objects. Regardless how many parts an object can be broken into, every part has a meaning. We hardly describe the world as a set of points, patterns or other low-level primitives. The abstract understanding is also key to our decision making. Therefore, to build a meaningful representation of the real world such that both machines and humans can understand and interact with, the semantic knowledge is a crucial key.

In computer vision and robotics, visual mapping and tracking has a long history of development, where many solutions have been developed for different types of sensors. In recent years, the RGB-D cameras are introduced to the computer science and robotic community, which measure the appearance and the geometry simultaneously in real-time and produce synchronized color and depth videos. The increasing availability of the commodity RGB-D cameras inspires many novel solutions to track the camera motion, to reconstruct the indoor environment and to interact with the physical world.

Another revolutionary breakthrough in the computer vision and machine learning field, comes from the rapid development of the deep learning algorithms with big data. Shown in abundant works, the deep neural network provides a powerful model that is capable to learn various sophisticated tasks, from classification, regression, to generative model predictions. In particular for image processing, algorithms developed upon the deep convolutional neural networks (CNNs) outperform many traditional model-based algorithms. For the task of visual semantic understanding, the CNN-based methods hold the state-of-the-art performances for almost every benchmark.

In this thesis, we investigate algorithms for semantic mapping and tracking based on the RGB-D vision. Our objective is to bridge the gap between the traditional dense mapping and tracking algorithms that rely on direct image methods and low-level features, to the emerging deep learning techniques that infer and aggregate high-level semantic understandings. Towards this goal, we focus on i) algorithms to obtain high-level abstracts and semantic understandings from RGB-D data and ii) methods to integrate such information to assist dense mapping, improve motion estimation and produce meaningful scene representations.

1.2 Background and Related Works

In this section, we provide an overview of the related works. Three research topics are covered. Firstly, we discuss the dense mapping techniques, with a focus on large-scale dense indoor reconstruction. Secondly, we overview the visual mapping and tracking algorithms, with a focus on the algorithms developed for RGB-D vision. Last, we consider the deep learning methods for visual understandings, with a focus on the convolutional neural networks for semantic segmentation.

1.2.1 Dense Large-scale Scene Reconstruction

Dense 3D reconstruction is a classical research topic in computer vision and graphics. With years of development, many algorithms have been proposed to tackle the different input data, the diverse output format and the scale of reconstruction. Consider RGB-D cameras capture the dense geometry in real time, here we are interested in methods suitable for large-scale scene reconstruction that are capable of processing video sequences and are robust to the noise from data acquisition and errors in pose estimation.

Volumetric Approaches Particularly for RGB-D mapping, the volumetric representation with signed distance function (SDF) is a commonly used method. Developed by **Curless and Levoy** (1996), this method integrates multi-view depth measurements into a 3D volume, where each voxel stores its signed distance to the surface. As a result, the 3D volume implicitly describes the 3D shapes with the level sets, and encodes the actual surface by the zero-level iso-surface. To extract the surface into meshes, the marching cubes algorithm (**Lorensen and Cline** 1987) can be applied. The volumetric mapping has a high demand for the memory usage, which is a critical bottleneck for the large-scale indoor mapping on the GPUs. To address this problem, **Steinbruecker et al.** (2013) develop an multi-resolution mapping solution using the octree. **Niessner et al.** (2013) develop an efficient volume compress based on the hashing technique.

Mesh from Dense Point Clouds Many RGB-D mapping algorithms reconstruct the indoor scenes into dense point cloud. Since points do not have volume, and point clouds cannot infer occlusions or connectivity, the mesh reconstruction techniques are often applied to better describe the underlying surfaces. Algorithms such as, the Delaunay triangulation (**Domiter and Zalik** 2008; **Fortune** 1997), the ball-pivoting (**Bernardini et al.** 1999), and the greed projection triangulation (**Gopi and Krishnan** 2002; **Marton et al.** 2009) faithfully reconstruct the geometry with different strategies to connect neighboring points. Due to the missing surface measurements, these algorithms typically contain holes. When the watertight reconstruction is desired, the Poisson surface reconstruction algorithm (**Kazhdan et al.** 2006; **Kazhdan and Hoppe** 2013) provides good approximations, which also de-noises the input point clouds.

Semantic Mapping In the attempt to produce semantic maps, **Hermans et al.** (2014) propose to train a random forest classifier to predict semantic RGB-D segmentation. The individual frame segmentations are aggregated into a 3D volume to be smoothed by a fully connected CRF. A consistent mapping is obtained with Bayesian fusion. Similarly, **Stückler et al.** (2015) proposed a probabilistic fusion of the semantic segmentations from individual RGB-D images from a random forest and integrate the predictions into multi-resolution voxel maps. **Armeni et al.** (2016) proposed a hierarchical parsing method for large-scale 3D point clouds of indoor environments. They first separate point clouds into disjoint spaces, *i.e.,* single rooms, and further cluster points at the object level with the help of handcrafted features. Another closely related technique is developed by **Häne et al.** (2017), where class-specified semantic priors are used to optimize semantic outdoor reconstruction from noisy input.

1.2.2 Simultaneous Mapping and Tracking

In computer vision and robotics, a closely related research to infer 3D geometry and camera motion from a sequence of 2D images is the technique of simultaneous localization and mapping (SLAM), also known as structure from motion (SfM). SLAM is a core technique for robotics, which also have many applications in reconstruction, augmented/virtual reality. The essential problem SLAM attempts to solve is, given an moving agent equipped with sensors to measure an unknown environment, build a map of the scene from the sensor data and localize the agent onto the map at the same time. The mapping and localization is intertwined. To localize, a map is required, and to update the map, localization is needed. Many sensors that measures the environments be used in a SLAM system, *e.g.,* radar, lidar, and cameras. In this section, we limit the discussions to camera-based visual SLAM. There are three subproblems within visual SLAM: camera tracking, map update and global optimization. Following we first review visual SLAM algorithms in general, then give a detailed discussion on RGB-D SLAM, and finally review dense mapping techniques.

1.2.2.1 Visual SLAM in General

In literature, many different cameras have be used to develop visual SLAM algorithms, including monocular, stereo and RGB-D. Independent of the cameras being used, following we classify visual SLAM algorithms with respect to how tracking is formulated.

Frame-based, Keyframe-based and Model-based Tracking Depending on how the individual frames are tracked, SLAM algorithms can be classified into: frame-to-frame tracking, frame-to-keyframe tracking, and frame-to-model tracking. The frame-to-frame tracking methods estimate the current camera motion with respect to the previous frame. This methods accumulate errors from all previous frames, which is the problem known as tracking drift. To reduce the problem, an better alternative is to employ the frame-to-keyframe tracking. In this scenario, keyframes are occasionally produced along the sequential tracking. The motion of the current frame is estimated with respect to the last keyframe instead of the last frame. Since every frame is tracked independently towards a keyframe, drifts are only accumulated between keyframes and not for the regular frames. The third option is the frame-to-model tracking, where the current frame is compared to the global model reconstructed given existing measurements. Given an accurate map, the frame-to-model tracking performs well, because every frame is tracked independently towards the model without accumulating drift. However, it is often expensive to construct an accurate model. Additionally, with tracking and mapping intertwined in SLAM algorithms, the tracking errors degrades the mapping performance and vice versa. Consequently, frame-to-model tracking is sensitive to small intermediate error, which can be magnified by the tracking-mapping feedback loop.

Direct versus Indirect One aspect distinguishes the different SLAM algorithms is the direct methods versus the indirect methods. With indirect methods, feature points and feature descriptors are used to establish correspondences for tracking. For example, the ORB SLAM algorithm (**Mur-Artal et al.** 2015) is an indirect method based on the ORB features (**Rublee et al.** 2011). On the contrast, the direct SLAM methods formulate camera tracking into image alignment with the raw data. Direct methods consider images as continuous 2D domain, where the neighborhood information provides important regularization, in particular for monocular and stereo vision to estimate geometry. Some classic direct SLAM algorithms include DTAM (Newcombe et al. 2011b), LSD SLAM (Engel et al. 2014), DSO SLAM (Engel et al. 2017), KinectFusion (Newcombe et al. 2011a) and DVO SLAM (Kerl et al. 2013a).

Sparse versus Dense Another distinction among different SLAM algorithms is characterized by the sparse methods versus the dense methods. The sparse SLAM algorithms use limited images points to perform tracking and consequently the reconstructed maps only represent sparse geometry. These sparse points are either robust feature points as in ORB SLAM (Mur-Artal et al. 2015) or sparsely sampled image pixels as in DSO SLAM (Engel et al. 2017). Sparse SLAM methods are commonly applied for monocular and stereo vision, where dense depth estimation is difficult and expensive. Dense methods attempt to recover the full 3D scene geometry for every 2D pixel. To this end, Newcombe et al. (2011b) proposed the DTAM method, where dense depth estimation is achieved based on the small baseline from consecutive monocular images. Engel et al. (2014) developed the LSD SLAM, where camera tracking and inverse depth prediction are jointly formulated within a direct tracking framework. Consequently LSD SLAM predicts semi-dense depth for monocular sequences. With the RGB-D cameras, geometry measurements are directly available, which motivates the development of many model-based SLAM methods (Dai et al. 2017b; Newcombe et al. 2011a; Whelan et al. 2016; Whelan et al. 2015b). Following we review these dense RGB-D SLAM methods in more details.

1.2.2.2 Canonical RGB-D SLAM Methods

Most existing RGB-D SLAM algorithms are direct dense methods with either keyframe-based tracking or predictive model-based tracking. One key difference between RGB-D SLAM in comparison to the monocular and the stereo SLAM is the available dense depth measurements. Despite the fact that depth images contain missing measurements due to sensor limitations, depth estimation is no longer needed and consequently searching for correspondences is easier. However, new challenges are introduced to RGB-D SLAM. In addition to sensor noise and limited depth range, efficient mapping algorithms are required to integrate dense depth images. Following, we review RGB-D SLAM algorithms given different map representations.

Volumetric Map Representation KinectFusion developed by Newcombe et al. (2011a) is considered to be the first RGB-D SLAM algorithm, which successfully demonstrates real-time dense tracking and mapping from a single moving depth camera. KinectFusion integrates depth images into a volumetric representation using the signed distance function (Curless and Levoy 1996). A predictive frame-to-model tracking is developed, where the ICP algorithm (Besl and McKay 1992) is used to align the current depth to the de-noised depth obtained from raycasting the SDF volume. KinectFusion is further studied and extended by many researchers. One of these extensions, known as Kintinuous by Whelan et al. (2012) develop the rolling volume technique to map unlimited space with KinectFusion. Kintinuous also uses a volumetric color fusion to obtain colored mapping. Whelan et al. (2015b) combines the visual odometry algorithm developed by Steinbruecker et al. (2011) with ICP to improve the tracking accuracy. They further apply the iSAM graph optimization (Kaess et al. 2008) to correct tracking errors and use spatial deformation technique (Sumner et al. 2007) to optimize mapping. Another extension is proposed by Bylow et al. (2013), where camera tracking is formulated directly on SDF for efficient computation. In KinectFusion and its extensions, errors in fused volumetric mapping can be magnified with a intertwined tracking and degrade subsequent SLAM performance. To address this issue, Dai et al. (2017b) propose the BundleFusion algorithm, where the voxel hashing technique (Niessner et al. 2013) is employed for efficient volumetric integration and de-integration.

Surfels Maps One limitation of KinectFusion alike RGB-D SLAM algorithms is the extensive memory consumption on GPU to store the volumetric data. An alternative map representation is surfels (Pfister et al. 2000), which can be considered as extended 3D points with attributes as location, orientation, dimension and color. Surfel maps was first applied in RGB-D SLAM algorithms by Henry et al. (2012) to map large-scale environment with a keyframe-based SLAM. For robust tracking, sparse visual features are combined with ICP. With a keyframe-based tracking, Henry et al. (2012) also perform loop closure detection and pose graph optimization to achieve global consistency. In this work, surfels are used as a representation for 3D reconstruction. To use surfels for online tracking, Niessner et al. (2013) develop a multi-resolution surfel registration algorithm with a probabilistic formulation. An alternative tracking with surfels is developed in Whelan et al. (2016), which predicts model depth and color to perform image alignment by rendering surfels.

Graph-based Representation: Decouple Tracking from Dense Mapping Most aforementioned SLAM algorithms attempt to perform tracking and dense mapping simultaneously, so as to perform predictive tracking. To obtain real-time tracking, GPU is used to parallel the intensive computations. Kerl et al. (2013a) proposed the DVO SLAM to decouple dense mapping from tracking. With a keyframe-based tracking, DVO SLAM represents maps as a pose graph consisting of all keyframes, which achieves real-time motion estimation and global optimization in real-time on a single threaded CPU. The DVO SLAM also combines the visual odometry method (Steinbruecker et al. 2011) with the direct depth alignment into a robust least square minimization and yield robust tracking. There are two major advantages to decouple dense mapping from tracking. First, it is capable of real-time CPU processing, and hence applicable to resource limited platform, *e.g.*, quadcoptors. Second, with a light-weight graph-based map representation, it is easier to correct tracking and mapping errors with a well-defined graph optimization framework (Grisetti et al. 2010). Note that as long as the original RGB-D sequence is available, it is always possible to obtain dense mapping at a post-processing step, for example using volumetric reconstruction algorithms (Curless and Levoy 1996; Steinbruecker et al. 2013). Fully optimized camera trajectory also yields better consistent dense mapping.

1.2.2.3 Integrating Geometric Primitives and Semantics

Geometric Primitives Several attempts have been made to integrate primitives into RGB-D SLAM. **Dou et al.** (2012) propose a SLAM method to combine planes and points of interests in frame-to-frame matching and use bundle adjustment (**Triggs et al.** 1999) to optimize camera poses, plane estimation and feature points location. For plane detection, a Hough-voting scheme is developed. The planes between frames are associated by RANSAC and the plane correspondences are tracked throughout the sequence. Since there may exist several tracks of the same plane entity, the planes are merged according to distance criteria. **Taguchi et al.** (2013) also exploit interest points and planes to formulate a RGB-D SLAM algorithm. They study various combinations of points and planes and use RANSAC to determine correspondences and estimates camera poses. **Trevor et al.** (2012) use RANSAC to find the dominant planes from dense RGB-D mapping and 2D laser measurements. SLAM is performed with an extended Kalman filter (EKF) framework. **Salas-Moreno et al.** (2014) integrate plane detection into surfel maps (**Keller et al.** 2013). By identifying planar surfels, they compute geometry alignment error directly as point to plane distance.

Semantic Templates Towards semantic RGB-D SLAM, **Salas-Moreno et al.** (2013) develop the SLAM++ algorithm to track and map at the object instance level. Given a predefined object template, the SLAM++ algorithm recognizes the same objects in the scene and further establishes constraints the between multiple observations of the same objects. These constraints together with the matched object templates are used in the global optimization to reduce the tracking error and to obtained the consistent reconstruction using the known templates. Such method is limited to indoor scenes with many identical objects. In addition, this method requires a predefined template database.

1.2.3 Convolutional Neural Networks for Semantic Scene Understanding

The topic of semantic scene understanding extends many dimensions. Here we limit the discussions to the semantic image segmentation, instance image segmentation and 3D scene parsing. Semantic image segmentation is the problem to predict a semantic label for every image pixel. It can be considered as the dense classification at the pixel level. Instance segmentation, takes on step further and predicts for every pixel the semantic meaning as well as the object index. Instance segmentation therefore can be considered as a combination of object detection and semantic segmentation. The problem of 3D scene parsing is an extension of the 2D image segmentation to 3D modeling. As opposed to images, the forms of 3D models are very diverse, ranging from unorganized point clouds, to surface manifolds embedded into meshes, and to structured volumetric representations.

In recent years, deep neural networks have shown superior performances for many learning tasks. With respect to learning from images, methods based on the state-of-the-art deep CNNs (**K. He et al.** 2016a; **Krizhevsky et al.** 2012; **Simonyan and Zisserman** 2015; **Szegedy et al.** 2015) continue to outperform the canonical model-based algorithms and hand-crafted features. Following, we overview CNN methods for scene understanding.

1.2.3.1 Semantic Instance Segmentation

Learning Semantic Features The early studies on CNNs have a focus on image classification. This is because CNNs increase the receptive field by drastically reducing the spatial resolution as the layers get deeper. While it is easy to downsample the input by pooling or strided convolutions, it is not straight-forward to learn the upsampling and train CNNs to for dense pixelwise predictions. One early attempt is proposed by **Couprie et al.** (2013), where a three-stage CNN is trained on the multi-scale Laplacian image pyramid to produce feature maps at different resolution. To achieve the final predictions, the features at all scale are aggregated according to the superpixel image segmentation. To process RGB-D images, the depth and color images are directly stacked into a four-channel input. **Gupta et al.** (2014) propose a differe solution based on the R-CNN object detection network (**Girshick et al.** 2014). The R-CNN extracts features from object region proposals, and further applies the linear support vector machines (SVM) for classification. To obtain the semantic segmentation, the object detection is further processed by the foreground-background superpixel-based segmentation. **Gupta et al.** (2014) extends the R-CNN method on RGB-D input. To this end, they propose to convert the depth image into a 3-channel image-alike HHA embedding, which encodes the horizontal disparity, the height above the ground, and the angle with the gravity.

End-to-End Semantic Segmentation An pioneering work in end-to-end semantic segmentation with CNNs is proposed by **Long et al.** (2015) with fully convolutional networks (FCNs). Using the transposed convolution introduced by **M. D. Zeiler et al.** (2010) to learn upsampling, FCNs produce a higher-resolution prediction from the coarse feature maps of $1/32$ input resolution. Inspired by FCNs and the auto-encoder CNNs (**Bengio et al.** 2007), the encoder-decoder CNNs are proposed concurrently by **Badrinarayanan et al.** (2017) and **Noh et al.** (2015) to achieve the prediction of the same resolution as the input. These works design the encoder based on the VGGNet (**Simonyan and Zisserman** 2015) for feature extraction, and mirror the encoder in decoder for upsampling. To reverse the pooling operation, the memorized unpooling proposed in **M. D. Zeiler et al.** (2011) are used. With the success of the very deep CNNs (**K. He et al.** 2016a; **K. He et al.** 2016b), keeping the decoder as the exact mirror of the encoder unnecessarily complicates the network design. To solve this problem, several CNNs architectures are developed. **Laina et al.** (2016) develop the up-projection layers for upsampling which effectively implements the upconvolution proposed by **Dosovitskiy et al.** (2015b). **G. Lin et al.** (2017) propose a multi-resolution refinement. **Pohlen et al.** (2017) utilize two CNN stream, where a full-resolution feature map is always maintained by one stream.

There also exist many solutions besides encoder-decoder CNNs. **Eigen and Fergus** (2015) concatenate multi-resolution CNNs to progressively refines the output resolution. **L.-C. Chen et al.** (2018); **L.-C. Chen et al.** (2015) adopt the dilated convolution to output a median-resolution prediction, and obtain a full-resolution output by post-processing with CRFs. **G. Lin et al.** (2016a); **G. Lin et al.** (2016b) formulate CNNs to learn the unary and pairwise potentials of a CRF model with piecewise approximation, which are used in the post-processing inference stage to obtain full-resolution predictions.

Instance Segmentations Beyond the semantic segmentation, impressive progress has been made in instance understandings with CNNs. One state-of-the-art CNN methods is the Mask RCNN algorithm developed by **K. He et al.** (2017). Mask RCNN is heavily built upon the Faster RCNN by **Ren et al.** (2017), which performs fast object detection. More specifically, the Faster RCNN introduces a region proposal network to propose object candidates in the form of bounding boxes. The method then extracts features from each bounding box for object classification and precise bounding box regression. Mask RCNN adopts this procedure, and further add a binary mask prediction for each bounding box to perform segmentation. To decouple the mask prediction from the category prediction, masks are generated for every class without competing, while the final prediction is decided by the object classification. Since the core to Mask RCNN is object detection, the instance information are naturally obtained. One limitation of the Mask RCNN is that not every pixels are assigned with predictions.

1.2.3.2 Semantic Scene Parsing

In contrast to the abundant studies of CNNs for image processing, it is less explored to apply CNNs for 3D data due the lack of structures in most 3D representations. The early works apply CNNs on volumetric data to perform object classification (**Wu et al.** 2015). The critical issue of the volumetric CNNs is the high memory consumption which limits the resolution. To reduce the problem, **Riegler et al.** (2016) develop the OctNet, which enables CNNs on a sparse octree to perform semantic segmentation on voxels. Recently **Song et al.** (2017) proposed a CNN method to jointly classify each voxel and predict the occupancy, which consequently produce a semantic volumetric mapping given a single view depth input. Besides the volumetric representation, **Qi et al.** (2017a); **Qi et al.** (2017b) develop the PointNet and PointNet++ to learn directly from point clouds. The key to these works is to adopt the K-nearest neighbor to structure the data for convolutional layers, and apply pooling to aggregate information.

1.2.3.3 Large-scale Semantic RGB-D Dataset

Large-scale semantic ground truth play a very important role in training CNNs and for the development of any data-hungry algorithms. To facilitate semantic understandings from RGB-D vision, many researchers dedicate great efforts to collect large-scale RGB-D dataset and develop algorithms to generate ground-truth annotations annotate these dataset. Following, we summarize the existing benchmarks.

Single-view Images NYUv2 developed by Silberman et al. (2012) is one of the early available RGB-D dataset. It comprises 1,449 RGB-D images from 464 captures of 26 different scenes types. Based on crowd sourcing using Amazon Machanical Turk (MTurk), dense pixelwise labeling is obtained. In addition to the annotations, NYUv2 also inpaint the raw depth images to provide ground-truth depth. Similar to NYUv2, the SUN RGB-D dataset is introduced by Song et al. (2015), which provide 10,335 images by crow sourcing on MTurk. SUN RGB-D obtains ground truth using LabelMe-style annotation (Russell et al. 2008), which draws 2D polygons on images and places 3D bounding boxes on point clouds to label objects. Consequently, SUN RGB-D provides 146,617 2D polygons and 64,593 3D bounding boxes in addition to densely annotated images. SUN RGB-D captures many indoor scenes and also includes data from NYUv2, Berkeley B3DO (Janoch et al. 2011) and SUN3D (Zeisl et al. 2013). Four RGB-D cameras are used in data acquisition, including Microsoft Kinect v1 and v2, Asus Xtion Pro and Intel RealSense.

Multi-view Sequence Moving towards RGB-D sequences and dense reconstruction, Hua et al. (2016) introduce the SceneNN dataset. SceneNN provides 100 optimized mesh reconstruction of indoor scenes, where each mesh is fully annotated at vertex level and subsequently projected onto 2D images to obtain fully labeled video sequence. SceneNN obtains the ground-truth label using the methods proposed in Nguyen et al. (2017), which 3D meshes are annotated first and 2D image labeling are obtained by projection. To perform mesh annotation, human interacts with mesh segments obtained from graph segmentation. To provide ground truth at large scale, ScanNet is introduced by Dai et al. (2017a), which collects 1513 fully annotated scenes with a total of 2.5 million views. ScanNet uses Bundle-Fusion (Dai et al. 2017b) to obtain dense reconstruction. Similar to SceneNN, ScanNet labels meshes from a graph-based segmentation and then projects the 3D annotations to achieve fully labeled videos.

Matterport Recently, the Matterport system are developed and used to collect large-scale indoor spaces. The Matterport consists of three RGB cameras and three depth cameras. In one capture, it rotates and stops six times to provide 360° 1280×1024 panorama RGB-D image registered from a total of 18 views. Matterport also provide mesh reconstruction. Matterport is first used by Armeni et al. (2017), where six indoor areas are collected and annotated. Also relying on Matterport cameras, Chang et al. (2017) further release the Matterport3D dataset with 90 building-scale scenes, where the screened Poisson algorithm (Kazhdan and Hoppe 2013) is used to obtain the mesh reconstruction. For ground-truth annotation, the large space is broken down into the scale of single rooms and then labeled with the similar methods used in ScanNet.

CHAPTER 2

Contributions

THE goal of this thesis is to incorporate semantic understanding into RGB-D mapping and tracking algorithms. To this end, we have focused on the following two perspectives. First, we have studied algorithms that extract higher-level abstracts and semantic understandings from RGB-D data. Second, we have explored methods that utilize higher-level understandings to optimize dense reconstruction, to enhance motion estimation, and to obtain meaningful map representations.

2.1 Major Constructions

This thesis develops novel algorithms to incorporate semantics into RGB-D mapping and tracking. The cumulative contents[1] comprise five publications and one submission, including **Ma et al.** (2013b), **Ma et al.** (2016), **Hazirbas et al.** (2017), **Ma et al.** (2017), **Ma et al.** (2018b) and **Ma et al.** (2018a)). Additionally, this thesis also presents supplementary material[2] based on four publications, including **Ma et al.** (2013c), **Ma et al.** (2013a), **Ma et al.** (2015), and **Whelan et al.** (2015a). These publications are the joint work with Thomas Whelan, Raphale Favier, Luat Do, Egor Bondarev, Peter H. N. de With, John McDonald, Christian Kerl, Jörg Stückler, Daniel Cremers, Caner Hazirbas, Csaba Domokos, Tao Wu, Julian Straub, Yufan Chen, Rohan Chabra, and Richard A. Newcombe. Eight of these works are published in highly ranked, peer reviewed conferences or journals. **Ma et al.** (2018b) is published in open access e-prints and **Ma et al.** (2018a) has been submitted to ACM 2018 SIGGRAPH ASIA. Table 2.1 presents an overview of all the publications that contribute to this thesis. The complete list of our publications is given in the bibliography. Overall, the contributions of this thesis are summarized into the followings.

[1] The *cumulative content* is a compilation of our publications (manuscripts) with minimal edition to fit the thesis.
[2] The *supplementary materials* are based on our publications to provide additional analysis.

Table 2.1: Our publications that contribute to this thesis, listed in the chronological order. The cumulative content (Part II) is a compilation of six publications marked in black. The supplementary analysis (Part III) are based on the publications marked in gray. A detailed contribution disclaimer is provided in Appendix.

Plane Segmentation and Decimation of Point Clouds for 3D Environment Reconstruction. Lingni Ma, Raphale Favier, Luat Do, Egor Bondarev and Peter H. N. de With. In proc. of IEEE 10th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 2013, pp. 43-49. ISSN: 2331-9852, DOI: 10.1109/CCNC.2013.6488423.

3D Colored Model Generation Based on Multiview Textures and Triangular Mesh. Lingni Ma, Luat Do, Egor Bondarev and Peter H. N. de With. In proc. of Seventh International Conference on Distributed Smart Cameras (ICDSC), Palm Springs, 2013, pp. 1-6. ISBN: 978-1-4799-2164-5, DOI: 10.1109/ICDSC.2013.6778206.

Planar Simplification and Texturing of Dense Point Cloud Maps. Lingni Ma, Thomas Whelan, Egor Bondarev, Peter H. N. de With and John McDonald. In proc. of European Conference on Mobile Robots (ECMR), Barcelona, Spain, 2013, pp. 164-171. ISBN: 978-1-4799-0263-7, DOI: 10.1109/ECMR.2013.6698837.

Multi-volume Mapping and Tracking for Real-time RGB-D Sensing. Lingni Ma, Egor Bondarev and Peter H. N. de With. In SPIE Image Processing: Algorithms and Systems XIII, San Francisco, 2015, vol. 9399, pp. 1-8. DOI: 10.1117/12.2083350.

Incremental and Batch Planar Simplification of Dense Point Cloud Maps. Thomas Whelan, Lingni Ma, Egor Bondarev, Peter H. N. de With and John McDonald. In Robotics Autonomous Systems. volume 69, issue C, (July 2015), pp. 3-4. ISSN: 0921-8890, DOI: 10.1016/j.robot.2014.08.019.

CPA-SLAM: Consistent Plane-model Alignment for Direct RGB-D SLAM. Lingni Ma, Christian Kerl, Jörg Stückler and Daniel Cremers. In proc. of IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016, pp. 1285-1291. ISBN: 978-1-4673-8026-3, DOI: 10.1109/ICRA.2016.7487260.

FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture. Caner Hazirbas, Lingni Ma, Csaba Domokos and Daniel Cremers. In proc. of Asian Conference on Computer Vision (ACCV), Springer, Cham, 2016, pp. 213-228. ISBN: 978-3-319-54181-5, DOI: 10.1007/978-3-319-54181-5_14.

Multi-view Deep Learning for Consistent Semantic Mapping with RGB-D Cameras. Lingni Ma, Jörg Stückler, Christian Kerl and Daniel Cremers. In proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 2017, pp. 598-605. ISSN: 2331-9852, ISBN: 978-1-5386-2682-5, DOI: 10.1109/IROS.2017.8202213.

Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform. Lingni Ma, Jörg Stückler, Tao Wu and Daniel Cremers. In Arxiv preprint: ARXIV:1808.01834 [cs.CV], 2018.

Human-in-the-loop Annotation for Large-scale 3D Semantic Datasets. Lingni Ma, Julian Straub, Yufan Chen, Thomas Whelan, Rohan Chabra, Richard A. Newcombe and Daniel Cremers. submitted to ACM SIGGRAPH ASIA 2018.

2.1.1 Dense Mapping with RGB-D Sequences

As an early work of this thesis, we explored algorithms to obtain photo-realistic dense 3D reconstructions from multi-view RGB-D data. In *"3D Colored Model Generation Based on Multiview Textures and Triangular Mesh"*, **Ma, Do, Bondarev** and **de With**, in proc. of Seventh International Conference on Distributed Smart Cameras (**Ma et al.** 2013a), we propose a novel algorithm to reconstruct high-resolution colored models for triangular meshes reconstructed from the multi-view RGB-D images. Instead of the direct texture mapping and texture blending, we propose to use 3D points to process and store color information. To this end, we develop techniques to perform adaptive 3D surface point upsampling and blend color using those points to store multi-view texture information. We further develop an algorithm to generate blended textures for textured model rendering. We demonstrate with experiments that our algorithms produce visually appealing high-resolution 3D reconstruction, while being robust to noises from data acquisition.

Another contribution we made is a novel multi-volume RGB-D mapping and tracking algorithm as proposed in *"Multi-volume Mapping and Tracking for Real-time RGB-D Sensing"*, **Ma, Bondarev** and **de With**, in SPIE Image Processing: Algorithms and Systems XIII (**Ma et al.** 2015). Inspired by the RGB-D SLAM algorithm KinectFusion **Newcombe et al.** (2011a) and its spatial extension Kintinuous **Whelan et al.** (2012), our multi-volume RGB-D SLAM algorithm deploys one small volume of high voxel resolution to obtain detailed maps of near-field objects, while utilizes another large volume of low voxel resolution to increase the robustness of tracking by including far-field scenes. We show with experiments that the proposed algorithm produces detailed reconstruction without sacrificing the tracking performance.

2.1.2 Optimize Tracking and Mapping with Plane Prior

Towards the goal of semantic mapping and tracking, we have explored geometric primitives, in particular planar surfaces. Planes are dominant features of the man-made environments. These surfaces often represent certain structural semantics, *e.g.*, walls, floors and ceilings. With many dense methods, these surfaces of simple geometry are yet over-represented by millions of points and polygons. In *"Plane Segmentation and Decimation of Point Clouds for 3D Environment Reconstruction"*, **Ma, Favier, Do, Bondarev** and **de With**, in proc. of IEEE 10th Consumer Communications and Networking Conference (**Ma et al.** 2013c), we develop an efficient plane detection algorithm based on the curvature to segment point cloud maps into planar and non-planar region. Together with the work of *"Planar Simplification and Texturing of Dense Point Cloud Maps"*, **Ma, Whelan, Bondarev, de With** and **McDonald**, in proc. of European Conference on Mobile Robots (**Ma et al.** 2013b), a hybrid mapping algorithm is proposed to reduce the redundancy and achieve a compact reconstruction. More specifically, we propose a quadtree-based algorithm to first decimate planar surface points and then triangulate the sparse data. We also develop a texture generation algorithm to preserve the visual information of the original dense input. Shown with experiments, our algorithm removes more than 90% planar points, leading to a triangulation with only 10% of the amount triangles that is otherwise required by the original dense data. The proposed decimation preserves the initial geometry and the texture generation method also provides visually appealing 3D maps. In *"Incremental and Batch Planar Simplification of Dense Point Cloud Maps"*, **Whelan, Ma, Bondarev, de With** and **McDonald**, in Robotics Autonomous Systems (**Whelan et al.** 2015a), we extend the proposed method to enable incremental batch processing for gradually expanding point cloud maps and integrate the algorithm into real-time RGB-D SLAM system.

Moving beyond mapping, in *"CPA-SLAM: Consistent Plane-model Alignment for Direct RGB-D SLAM"*, **Ma, Kerl, Stückler** and **Cremers**, in proc. of IEEE International Conference on Robotics and Automation (**Ma et al.** 2016), we propose a real-time keyframe-based RGB-D SLAM algorithm that consistently integrates plane priors into tracking and mapping. Our method represents the 3D scene with a global planar map, which are concurrently extracted from all the keyframes, and estimated in a global coordinates. We then formulate camera tracking into a mixture of the direct image alignment towards a keyframe and the model-based alignment towards the global plane model. We derive an equivalent probabilistic formulation to solve the motion estimation with expectation-maximization (EM). Furthermore, the planar map representation is used in a global graph optimization, where additional constrains are formulated between keyframes without overlapping views once a common plane is observed. With these additional constrains, we can further reduce the tracking errors and achieve better mapping consistency. We demonstrate with intensive experiments that our method obtains the state-of-the-art tracking accuracy on challenging benchmarks.

2.1.3 Learning Semantics from Multi-view RGB-D Data

Going beyond plane priors, this thesis also investigates semantic scene understanding in a broader perspective. To this end, we exploit the deep learning techniques, especially the convolutional neural networks (CNNs) for single-view semantic segmentation and for multi-view semantic mapping. In recent years, CNNs have attained a breakthrough in many computer vision tasks. However, with the RGB-D data, it is less explored how to incorporate the depth information into the CNNs learning. To tackle this problem, we propose a novel CNN architecture, namely FuseNet in *"FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture"*, **Hazirbas, Ma, Domokos** and **Cremers**, in proc. of Asian Conference on Computer Vision (**Hazirbas et al.** 2017). The designed network consists of two separate branches to filter color and depth images, while constantly fusing the depth features into the color features as the network goes deeper. To demonstrate the efficiency of FuseNet, we apply it to an encoder-decoder network and show that our method achieves competitive results on the SUN RGB-D semantic segmentation benchmark (**Song et al.** 2015).

Following FuseNet, we develop it further to learn consistent semantic mapping for multi-viwe RGB-D sequences with the work *"Multi-view Deep Learning for Consistent Semantic*

Mapping with RGB-D Cameras", **Ma, Stückler, Kerl** and **Cremers**, in proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (**Ma et al.** 2017). With the objective to enforce the multi-view consistency, we regularize the network training using the camera trajectories obtained by the SLAM algorithm. The principle is to warp network outputs of multiple viewpoints into a reference view (in correspondence with the keyframe in a SLAM setting) at different stages, where different approaches can be employed to learn the invariant features under viewpoint changes. Based on warping, our semi-supervised training method also alleviates the need for large amounts of annotated data which is typically expensive to obtain. Complementary to our training approach, we aggregate the predictions of the trained network in keyframes to increase the segmentation accuracy at inference time. The predictions of neighboring images are fused into the keyframe in a probabilistic way. Shown in experiments, the proposed end-to-end training achieves state-of-the-art performance on the NYUv2 dataset (**Silberman et al.** 2012) with single-view segmentation as well as with multi-view semantic fusion. While the fused keyframe segmentation can be directly used in robotic perception, our approach also provide a useful building block for semantic SLAM.

Semantic segmentation with CNNs typically uses a encoder-decoder network architecture to achieve dense pixel-wise prediction. This task consist of two concurrent goals: classification with the encoder and localization with the decoders. While the classification is well studied by many works, the design of decoder to obtain detailed prediction remains challenging. Motivated by the structural analogy between multi-resolution wavelet analysis and the pooling/unpooling operations of CNNs, we introduce WCNN, a novel method that integrates the discrete wavelet transform (DWT) into the encoder-decoder CNNs in *"Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform"*, **Ma, Stückler, Wu** and **Cremers**, in In Arxiv preprint: ARXIV:1808.01834 [cs.CV], (**Ma et al.** 2018b). In WCNN, DWT is performed at encoder to decompose the feature maps into frequency bands when the spatial resolution is reduced. The high-frequency components are then used jointly with the corresponding coarse feature maps at the decoder to upscale the resolution with inverse DWT. Based on DWT, we also developed two wavelet-based pyramids to bridge the encoder and decoder with global context information. These pyramids, successively reduce the spatial resolution to the finest scale and increase the receptive field to the entire input image by building a multi-resolution wavelet transform. Based on DWT/iDWT, the pyramids systematically upscale the global context to the desired resolution. Shown with experiments, WCNNs improve the fine-resolution prediction accuracy over the baseline CNNs and yield the state-of-the-art performance for the Cityscape dataset (**Cordts et al.** 2016).

2.1.4 Towards the Generation of Large-scale Semantic RGB-D Dataset

Large-scale ground truth dataset plays a crucial role in many computer vision and machine learning algorithm, particularly for data-hungry methods, such as deep learning. To this end, we propose a unified tool to obtain large-scale high-quality and geometrically consistent semantic 3D dataset in *"Human-in-the-loop Annotation for Large-scale 3D Seman-*

tic Datasets", **Ma, Straub, Chen, Whelan, Chabra, Newcombe** and **Cremers,** submitted to ACM SIGGRAPH ASIA 2018 (**Ma et al.** 2018a). The proposed annotation presents three advantages in comparison to the existing algorithms. First, we propose a segmentation-aided free-form mesh labeling method to ensure detailed and accurate labeling. Together with a novel tree structure to organize the semantic annotations into hierarchies, our tool naturally encodes the semantic and instance information, as well as the relationship between different semantic categories. Second, we develop a human-aided geometry completion technique, which enables human annotators to insert missing surfaces into the mesh reconstruction. This enhances the labeling consistency when transferring the 3D annotations into 2D images via rendering. With a post-processing to refine the rendered labels with a joint bilateral filter, our tool produces fully annotated RGB-D videos. Third, we propose a close-loop annotation scheme. We train instance semantic annotations with deep CNNs based on existing annotated data, and then aggregate the 2D network predictions into the mesh to bootstrap annotations of the unlabeled data. To exam the efficiency of the proposed algorithms, we show with experiments the developed algorithm achieve better labeling accuracy and 2D/3D consistency in comparison to the existing methods. Furthermore, we collect 20 indoor scenes to valid the close-loop scheme, where the proposed algorithm predicts instance segmentation for a large office areas with high accuracy.

2.2 Thesis Outline

This thesis is structured into four parts.

Part I presents the research problem and provides the supporting theoretical background. In detail, Chapter 1 defines the research problem and gives an overview of the state-of-the-art methods on 3D reconstruction, RGB-D SLAM and semantic visual understanding. Chapter 2 gives an overview the selected publications and summarizes the major contributions of this thesis. Chapter 3 provides the necessary technical background to support this work, including the fundamentals on mathematics, on RGB-D SLAM and on deep CNNs.

Part II provides the cumulative content as a compilation of six research publications. Starting with the simple geometry primitive, planes, Chapter 4 proposes a 3D reconstruction algorithm to efficiently decimate, triangulate and texture the dense colored point clouds by exploiting planar structures (**Ma et al.** 2013b). Chapter 5 proposes a novel real-time RGB-D SLAM algorithm to integrate planes into camera tracking, map representation, and global optimization (**Ma et al.** 2016). Moving towards semantic understanding, Chapter 6 introduces a novel CNN architecture FuseNet for RGB-D learning, which extracts complementary features from the depth and the color images with different branches and fuses them consistently as the network goes deeper (**Hazirbas et al.** 2017). Based on FuseNet, Chapter 7 further explores solutions to constrain training RGB-D sequences with multi-view consistency and achieves robust semantic segmentation (**Ma et al.** 2017). Chapter 8 introduces discrete wavelet transform into encoder-decoder CNNs and improves the details in dense pixelwise inference, *e.g.,* semantic prediction (**Ma et al.** 2018b). Last, Chapter 9 proposes an annotation algorithm and a close-loop labeling scheme to generate large-scale semantic ground-truth dataset (**Ma et al.** 2018a).

Part III presents the supplementary materials to Part II. The content originates from our research publications that are either early exploration or extension of the work presented in Part II. Chapter 10 discusses the dense RGB-D mapping algorithms to obtain visually appealing 3D reconstructions, based on the work of **Ma et al.** (2013a) and **Ma et al.** (2015). Chapter 11 describes the plane detection algorithm proposed in (**Ma et al.** 2013c), which Chapter 4 is built upon. This chapter also presents the extension of Chapter 4 into incrementally expanding point cloud maps based on **Whelan et al.** (2015a). Chapter 12 discusses an ablation study on network design of Chapter 7 and provides semantic mapping results.

In Part IV Chapter 13 concludes this thesis and discusses the future research directions.

CHAPTER 3

Theoretical Background

THIS chapter summarizes the mathematical theories that are used throughout the thesis. In addition, we provide the technical background for RGB-D mapping and tracking. Last, an overview of the deep convolutional neural networks (CNNs) is presented.

3.1 Mathematical Preliminaries

Notations In this section, we use x for a scalar value, $\mathbf{x} \in \mathbb{R}^k$ and $(\cdot)^T$ for a k dimensional vector, X and $[\cdot]$ for a matrix, and $\mathbf{X}$ and $\{\cdot\}$ for a set. The subscript is used to index data and the bracket superscripts are used to index iterations or time stamps.

3.1.1 Probability Theory

Consider two random variables $\mathbf{x}$ and $\mathbf{y}$, with the probability distribution $p(\mathbf{x})$, and $p(\mathbf{y})$. The conditional distribution of variable $\mathbf{x}$ conditioned on $\mathbf{y}$ is $p(\mathbf{x} \mid \mathbf{y})$, the joint distribution is $p(\mathbf{x}, \mathbf{y})$, and the marginal distribution $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$. The Bayesian rule yields

$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x})}{p(\mathbf{y})} = \frac{p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x})}{\sum_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{y})} . \tag{3.1}$$

3.1.1.1 Parameter Estimation

Consider a general statistical model, with k dimensional parameters $\boldsymbol{\theta} \in \mathbb{R}^k$ and a set of observations $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$. The *maximum likelihood* (ML) estimation is defined by

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p(\mathbf{X} \mid \boldsymbol{\theta}) . \tag{3.2}$$

The *maximum a posterior* (MAP) Estimation is defined by

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \mathbf{X}) = \arg\max_{\boldsymbol{\theta}} \frac{p(\mathbf{X} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathbf{X})} = \arg\max_{\boldsymbol{\theta}} p(\mathbf{X} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) . \tag{3.3}$$

The function $p(\mathbf{X} \mid \theta)$ is referred as the *likelihood* function, $p(\theta \mid \mathbf{X})$ as the posterior distribution and $p(\theta)$ as the *prior* distribution. Note that with a uniformly distributed prior, MAP estimation is reduced to ML estimation. Unless otherwise stated, the observation $\mathbf{X}$ is assumed to be independent and identically distributed (*i.i.d.* condition), hence,

$$\theta_{\mathrm{ML}} = \arg\max_{\theta} \prod_{i}^{n} p(x_i \mid \theta) = \arg\min_{\theta} \sum_{i}^{n} -\log p(x_i \mid \theta) \tag{3.4}$$

$$\theta_{\mathrm{MAP}} = \arg\max_{\theta} \prod_{i}^{n} p(x_i \mid \theta) p(\theta) = \arg\min_{\theta} \sum_{i}^{n} -\log p(x_i \mid \theta) - \log p(\theta)\ . \tag{3.5}$$

3.1.1.2 Multivariant Probability Distributions

Gaussian The k-dimensional Gaussian distribution takes the form

$$p(x \mid \mu, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{k/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (x - \mu) \right)\ , \tag{3.6}$$

where $\mu \in \mathbb{R}^k$ is the mean vector, and $\boldsymbol{\Sigma} \in \mathbb{R}^{k \times k}$ is the covariance matrix. The value $(x - \mu)^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (x - \mu)$ is the squared Mahalanobis distance. We now derive the ML estimation for the parameters. First note the following partial derivatives (the equality holds true given covariance matrix is positive semi-definite),

$$\frac{\partial}{\partial x} \left(x^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} x \right) = 2 \boldsymbol{\Sigma}^{-1} x\ , \tag{3.7}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \left(x^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} x \right) = -\boldsymbol{\Sigma}^{-1} x x^{\mathsf{T}} \boldsymbol{\Sigma}^{-1}\ , \tag{3.8}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} |\boldsymbol{\Sigma}| = |\boldsymbol{\Sigma}| \boldsymbol{\Sigma}^{-1}\ . \tag{3.9}$$

Using Equation (3.4), the ML estimation for Gaussian hence is,

$$\frac{\partial \log p(x \mid \mu, \boldsymbol{\Sigma})}{\partial \mu} = 0 \implies \sum_{i}^{n} \boldsymbol{\Sigma}^{-1} (x_i - \mu) = 0 \implies \mu_{\mathrm{ML}} = \frac{1}{n} \sum_{i}^{n} x_i\ . \tag{3.10}$$

$$\begin{aligned} \frac{\partial \log p(x \mid \mu, \boldsymbol{\Sigma})}{\partial \boldsymbol{\Sigma}} = 0 &\implies \sum_{i}^{n} \boldsymbol{\Sigma}^{-1} \left((x_i - \mu)(x_i - \mu)^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} - 1 \right) = 0 \\ &\implies \boldsymbol{\Sigma}_{\mathrm{ML}} = \frac{1}{n} \sum_{i}^{n} (x_i - \mu)(x_i - \mu)^{\mathsf{T}}\ . \end{aligned} \tag{3.11}$$

Student t-Distribution The k-dimensional student t-distribution takes the form

$$p(x \mid \mu, \boldsymbol{\Sigma}, \nu) = \frac{\Gamma(\nu/2 + k/2)}{\Gamma(\nu/2)(\pi\nu)^{k/2} |\boldsymbol{\Sigma}|^{1/2}} \left(1 + \frac{(x - \mu)^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (x - \mu)}{\nu} \right)^{-(\nu + k)/2}\ , \tag{3.12}$$

where $\Gamma(\cdot)$ is the Gamma function. The parameter ν is called the degrees of freedom. With $\nu = 1$, the t-distribution becomes Cauchy distribution, while with $\lim \nu \to \infty$, t-distribution amounts to Gaussian with mean μ and covariance $\boldsymbol{\Sigma}$. The t-distribution can be interpreted as an infinite mixture of Gaussian having the same mean but different variance, resulting in a heavy tailed distribution. Student t-distribution hence behaves as a generation of Gaussian, whose ML parameter estimation is robust to outliers. We direct readers to **Bishop** (2006) for detailed explanations.

Following the ML estimation procedure as with Gaussian for parameter μ and $\boldsymbol{\Sigma}$, yield

$$\frac{\partial p(x \mid \mu, \boldsymbol{\Sigma}, \nu)}{\partial \mu} = \sum_i^n \frac{\nu + k}{\nu + (x_i - \mu)^\mathsf{T} \boldsymbol{\Sigma}^{-1} (x_i - \mu)} (x_i - \mu) = 0 \,, \tag{3.13}$$

$$\frac{\partial p(x \mid \mu, \boldsymbol{\Sigma}, \nu)}{\partial \boldsymbol{\Sigma}} = \sum_i^n \frac{\nu + k}{\nu + (x_i - \mu)^\mathsf{T} \boldsymbol{\Sigma}^{-1} (x_i - \mu)} (x_i - \mu)(x_i - \mu)^\mathsf{T} \boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1} = 0 \,. \tag{3.14}$$

Compared to Gaussian, the closed-form solution does not exist for t-distribution. Setting

$$w_i = \sum_i^n \frac{\nu + k}{\nu + (x_i - \mu)^\mathsf{T} \boldsymbol{\Sigma}^{-1} (x_i - \mu)} \,, \tag{3.15}$$

the mean and covariance can be estimated iteratively

$$\mu^{t+1} = \frac{\sum_i^n w_i^{(t)} x_i}{\sum_i^n w_i^{(t)}} \quad , \quad \boldsymbol{\Sigma}^{t+1} = \frac{\sum_i^n w_i^{(t)} \left(x_i - \mu^{(t+1)}\right)\left(x_i - \mu^{(t+1)}\right)^\mathsf{T}}{n} \,. \tag{3.16}$$

Mixture Distribution and Expectation Maximization By taking linear combinations of the basic probability distribution, the mixture distributions can be formulated. In general, the mixture model consists of m components of distribution $p(x \mid \boldsymbol{\Theta})$ takes the following forms

$$p(x \mid \boldsymbol{\Theta}) = \sum_j^m \pi_j p(x \mid \theta_j) \,, \tag{3.17}$$

with the *mixing coefficients* π_j sum to one, and $\Theta = \{\theta_1, \theta_2, \cdots, \theta_m\}$. The log-likelihood function of a mixture model takes the form

$$\log p(x \mid \boldsymbol{\Theta}) = \log \prod_i^n \sum_j^m \pi_j p(x_i \mid \theta_j) = \sum_i^n \log \sum_j^m \pi_j p(x_i \mid \theta_j) \,, \tag{3.18}$$

which is different to optimize due the sum inside the log function. To to find the ML estimation for such probabilistic model, the iterative estimation method *expectation-maximization* (EM) provides a general solution. The EM algorithm assumes there is a latent variable $z \in \mathbb{R}^m$ for the observed data x, which takes the form of an one-hot indicator, *i.e.*, $\sum_j^m z_j = 1$ and satisfies,

$$p(z) = \prod_j^m \pi_j^{z_j} \,, \tag{3.19}$$

$$p(x \mid z_j = 1) = p(x \mid \theta_j) \implies p(x \mid z) = \prod_j^m p(x \mid \theta_j)^{z_j} \,. \tag{3.20}$$

With the latent variable, the likelihood distribution of the complete-data observation is given

$$p(x, z \mid \Theta) = \prod_{i}^{n} \prod_{j}^{m} \left(\pi_j p(x_i \mid \theta_j)\right)^{z_{ij}} , \tag{3.21}$$

which brings back the log-likelihood to the form that is significantly easier to optimize. In practice, the latent variable cannot be observed, but we can instead estimate the expected complete data distribution, with respect to the posterior $p(z \mid x, \Theta)$. Given that

$$p(z \mid x, \Theta) = \frac{p(x, z \mid \Theta)}{\sum_z p(x, z \mid \Theta)} \propto p(x, z \mid \Theta) , \tag{3.22}$$

The expectation of the indicator variable

$$\mathbb{E}[z_{ij}]_{\mid p(z \mid x, \Theta)} = \frac{\pi_j p(x_i \mid \theta_j)}{\sum_j^m \pi_j p(x_i \mid \theta_j)} := \gamma_{ij} , \tag{3.23}$$

where γ_{ij} also known as the *responsibility*. Combine Equation (3.21) and Equation (3.23), the expectation of the complete-data log-likelihood is thus,

$$\mathbb{E}\left[\log p(z, x \mid \Theta)\right]_{\mid p(z \mid x, \Theta)} = \sum_{i}^{n} \sum_{j}^{m} \gamma_{ij} \left(\log \pi_j + \log p(x_i \mid \theta_j)\right) . \tag{3.24}$$

Starting with some initial value of Θ and mixing coefficients π_j, the EM algorithm is an iteration of E-step and M-step. With E-step, the responsibilities γ_{ij} are evaluated given the current model parameters according to Equation (3.23). With M-step, parameters are re-estimated by ML estimation of Equation (3.24). The algorithm terminate when the estimation converges.

3.1.1.3 Bayesian Update

Consider a sequence of observations $\mathbf{X}^t = \{x_1, x_2, \cdots, x_t\}$ where the subscript t denotes the time stamp. Each observation contributes a probability prediction to the same random variable y with distribution $p(y_t \mid x_t)$. To obtain the fused probability distribution $p\left(\mathbf{Y}^t \mid \mathbf{X}^t\right)$, the Bayesian update (also known as Bayesian fusion) is formulated as follows.

$$p(\mathbf{Y}^t \mid \mathbf{X}^t) = \frac{p\left(\mathbf{X}^t \mid \mathbf{Y}^t\right) p(\mathbf{Y}^t)}{p(\mathbf{X}^t)} = \frac{p\left(x_t \mid \mathbf{X}^{t-1}, \mathbf{Y}^t\right) p(\mathbf{Y}^t \mid \mathbf{X}^{t-1}) p(\mathbf{X}^{t-1})}{p(x_t \mid \mathbf{X}^{t-1}) p(\mathbf{X}^{t-1})} . \tag{3.25}$$

Assuming observations satisfy the *i.i.d.* condition. Additionally, the fused prediction is smooth, thus $p\left(\mathbf{Y}^t \mid \mathbf{X}^{t-1}\right) \approx p\left(\mathbf{Y}^{t-1} \mid \mathbf{X}^{t-1}\right)$. Equation (3.25) can be further simplified into

$$p(\mathbf{Y}^t \mid \mathbf{X}^t) = \frac{p\left(x_t \mid \mathbf{Y}^t\right)}{p(x_t)} p(\mathbf{Y}^{t-1} \mid \mathbf{X}^{t-1}) = \frac{p\left(\mathbf{Y}^t \mid x_t\right) p(x_t)}{p(\mathbf{Y}^t) p(x_t)} p(\mathbf{Y}^{t-1} \mid \mathbf{X}^{t-1}) . \tag{3.26}$$

Assuming the prior $p(\mathbf{Y}^t)$ is constantly smooth, and hence can be viewed as a normalizing factor, Equation (3.26) can be reduced to the form

$$p(\mathbf{Y}^t \mid \mathbf{X}^t) = \frac{p(y_t \mid x_t)}{p(\mathbf{Y}^t} p(\mathbf{Y}^{t-1} \mid \mathbf{X}^{t-1}) = \frac{\prod_t p(y_t \mid x_t)}{\prod_t p(\mathbf{Y}^t)} . \tag{3.27}$$

Equation (3.27) is the Bayesian update formula, which can be interpreted as the fused prediction at time t is the normalized element-wise product of the each individual prediction $p(\mathbf{y}_t \mid \mathbf{x}_t)$.

3.1.1.4 KL-divergence

The *Kullback-Leibler (KL) divergence* is a function that measures the difference between two probability distributions $p(\mathbf{x})$ and $q(\mathbf{x})$. In the discrete settings, it is defined by

$$\mathrm{KL}(p \parallel q) = \sum_{\mathbf{x}} p(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} = \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}) - p(\mathbf{x}) \log q(\mathbf{x}) \,, \tag{3.28}$$

where $-p(\mathbf{x}) \log p(\mathbf{x})$ is the *entropy* of $p(\mathbf{x})$, and $-p(\mathbf{x}) \log q(\mathbf{x})$ is the *cross entropy* between $p(\mathbf{x})$ and $q(\mathbf{x})$. The KL divergence is none negative and only equals to zero if the two distributions are identical. It is also non-symmetric, where by convention, $p(\mathbf{x})$ is the true distribution and $q(\mathbf{x})$ is the noisy estimation.

3.1.2 Optimization

An optimization problem is defined by an objective function $f(\mathbf{x})$ (also referred as the cost function or the loss function for machine learning, and the energy function for variational methods) and constraints if any, the goal is find an optimizer such that the function value is at its global optimum. Unless otherwise stated, we formulate all optimization as minimization, given maximization can be converted by multiply negative one to the objective function.

Gradient, Jacobian and Hessian The stationary points of a function define the local minimum or maximum. The minimum of all local minimum is the global minima. Given this condition, an important step of optimization is to obtain the equation with the gradient equals zero. With this in mind, we define the following terms. Given a multi-variable functional $f(\mathbf{x}) : \mathbb{R}^k \mapsto \mathbb{R}$, the *gradient* ∇f is defined

$$\nabla f = (\nabla f_i)^T = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_i} \right)^T \quad \in \mathbb{R}^k \,, \tag{3.29}$$

which defines the steepest direction to increase the function value at a given evaluation point. For a vector-valued function, $f(\mathbf{x}) : \mathbb{R}^k \mapsto \mathbb{R}^m$, the gradient is generalized to the *Jacobian* matrix

$$\mathbf{J} = [J_{ij}] = \left[\frac{\partial f_i}{\partial x_j} \right] \quad \in \mathbb{R}^{m \times k} \,. \tag{3.30}$$

For functionals, the Jacobian of the gradient is the *Hessian* matrix $\mathbf{H}$, defined by

$$\mathbf{H} = [H_{ij}] = \left[\frac{\partial \nabla f_j}{\partial x_j} \right] = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right] \quad \in \mathbb{R}^{k \times k} \,, \tag{3.31}$$

which describes the local curvature of the function. Unless otherwise stated, the objective function in this thesis are all functionals.

Convexity When the objective function and the solution space are both convex, the local minimizer is also the global one. Unfortunately this condition is not valid for most problems in this thesis. Often non-convex optimizations are solved with iterative methods, where good initialization is crucial. With a bad initialization, the optimization converges slowly, converges to bad local minimum or even diverges. There are many advanced algorithms to relax the problem, which are outside the scope of this thesis.

3.1.2.1 Approximation for Nonlinear Optimization

Linearization Consider an objective function that is non-linear with respect to the optimizing variable, the gradient, Jacobian, and Hessian is not straight-forward to compute. A common practice is to linearize the optimization and approximate the solution iteratively. Starting with some initial guess, with iteration $(k+1)$, the goal is to find an incremental update $\Delta\mathbf{x}$ such that $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}$ is the minimizer of the $(k+1)$-th linearized objective function. Using the first-order Taylor expansion and omitting the higher order terms, the linearized objective function at $\mathbf{x}^{(k)}$ is described by

$$f_{\text{lin}}\left(\mathbf{x}^{(k+1)}\right) \approx f\left(\mathbf{x}^{(k)}\right) + \Delta\mathbf{x}^{\mathsf{T}}\nabla f\left(\mathbf{x}^{(k)}\right) . \tag{3.32}$$

Setting the derivation of f_{lin} to zero, yields the following equality for the minimizer,

$$\mathbf{H}\left(\mathbf{x}^{(k)}\right)\Delta\mathbf{x} = -\nabla f\left(\mathbf{x}^{(k)}\right) . \tag{3.33}$$

As to be show in the following text, this equality defines the update equation to solve for $\Delta\mathbf{x}$ for all the iterative methods used in this thesis. The distinction among different optimization methods mainly comes from how the Hessian is computed.

This thesis only considers functionals for objective function. For intermediate calculation, the linearization of vector-valued function $\mathbf{f}(\mathbf{x}) : \mathbb{R}^k \mapsto \mathbb{R}^m$ takes the following form

$$\mathbf{f}_{\text{lin}}(\mathbf{x}+\Delta\mathbf{x}) \approx \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\Delta\mathbf{x} , \tag{3.34}$$

with $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix defined in Equation (3.30) evaluated at $\mathbf{x}$.

Quadratic Approximation Similar to linear approximation, for functionals (at least $\mathbb{C}^2$ continuous) the local quadratic approximation can be obtained with the second-order Taylor expansion,

$$f_{\text{quad}}(\mathbf{x}^{(k+1)}) \approx f(\mathbf{x}^{(k)}) + \Delta\mathbf{x}^{\mathsf{T}}\nabla f\left(\mathbf{x}^{(k)}\right) + \frac{1}{2}\Delta\mathbf{x}^{\mathsf{T}}\mathbf{H}\left(\mathbf{x}^{(k)}\right)\Delta\mathbf{x} . \tag{3.35}$$

Equation (3.35) can be interpreted to approximate the gradient with $\nabla f\left(\mathbf{x}^{(k)}\right) + \mathbf{H}\left(\mathbf{x}^{(k)}\right)\Delta\mathbf{x}$. Hence, the equality of Equation (3.33) also holds true for quadratic approximation. When the function is locally quadratic, the second-order approximation yields much faster convergence. However, Hessian is usually very expensive to compute for most problems.

3.1.2.2 First-order Methods

Gradient Descent *Gradient descent* is the most common optimization method, where minimization is achieved by iteratively moving in the negative gradient direction,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \lambda \nabla f\left(\mathbf{x}^{(k)}\right) , \tag{3.36}$$

where the parameter λ defines the step size, also known as the *learning rate* in deep learning (see Section 3.3). Compare to Equation (3.33), gradient descent amounts to approximate the Hessian matrix with an arbitrary step size $\mathbf{H} \approx \eta^{-1}\mathbf{I}$ with $\lambda = \eta^{-1}$. Despite the gradient defines the steepest direction for increase, the optimal step size for updating is unknown. Hence, gradient descent suffers from slow convergence and the zig-zagging optimization pattern.

Mini-batch Gradient Descent and Stochastic Gradient Descent One variant of the gradient decent optimization is the *mini-batch gradient descent*. In large-scale applications, *e.g.,* training deep neural networks with millions of images, evaluation of the gradient given the entire training data is very expensive and often unnecessary to perform one update iteration. In practice, the gradient is aggregated over a relatively small batch of data, which is randomly sampled each iteration. For example, a batch of 256 images is the typical size to train ResNet (**K. He et al.** 2016a) with ImageNet (**Krizhevsky et al.** 2012), which provides a total number of 1.2 million training samples. This method is referred as mini-batch gradient descent. Such method works because of the correlation between the data, where the data distribution of a small batch size is a good approximation to the distribution given the entire data, hence provide a good approximation to the gradient. An extreme case of mini-batch gradient descent is to set the batch size to one, which is known as *stochastic gradient descent* (SGD). With a slight abuse of terminology, we use mini-batch gradient descent and SGD interchangeably.

Momentum In practice when applying SGD to train neural networks, *momentum* μ is used to speed up the training process and to avoid stucking at local minimum (**Bottou** 2012). SGD with momentum use the following updates,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \lambda \nabla f\left(\mathbf{x}^{(k)}\right) + \mu \mathbf{x}^{(k)} . \tag{3.37}$$

Gradient-based Methods for Neural Networks There exists many gradient-based optimization methods developed to train deep neural networks, including AdaDelta by **M. D. Zeiler** (2012), AdaGrad by **Duchi et al.** (2011), Adam by **Kingma and Ba** (2015), Nesterov by **Sutskever et al.** (2013), and RMSprop by **Tieleman and Hinton** (2012). Among these methods, Adam is the more popular one, which computes individual learning rates adaptively for different parameters using estimates of first and second moments of the gradients. It combines the advantages of both AdaGrad and RMSprop. For each optimizing variable

$x_i \in \mathbb{R}$ at iteration $(k+1)$, Adam estimates the first moment (mean) $m_i^{(k+1)}$ and the second raw moment (biased variance) $v_i^{(k+1)}$ of the gradient with moving average,

$$m_i^{(k+1)} = \beta_1 m_i^{(k)} + (1-\beta_1)\nabla f\left(x_i^{(k)}\right) \ , \qquad \hat{m}_i^{(k+1)} = m_i^{(k+1)} / \left(1-\beta_1^{k+1}\right) \ , \tag{3.38}$$

$$v_i^{(k+1)} = \beta_2 v_i^{(k)} + (1-\beta_2)\nabla f\left(x_i^{(k)}\right)^2 \ , \qquad \hat{v}_i^{(k+1)} = v_i^{(k+1)} / \left(1-\beta_2^{k+1}\right) \ , \tag{3.39}$$

where the two hyper-parameters $\beta_1, \beta_2 \in [0,1)$ control the decay rate. Notice that $\hat{m}_i^{(k+1)}$ and $\hat{v}_i^{(k+1)}$ are the unbiased moment estimation. Given a base learning rate λ, Equation (3.38) and Equation (3.39), the update step for each parameter is defined

$$x_i^{(k+1)} = x_i^{(k)} - \lambda \frac{\hat{m}_i^{(k)}}{\sqrt{\hat{v}_i^{(k)}} + \epsilon} = x_i^{(k)} - \lambda \frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \frac{m_i^{(k)}}{\sqrt{v_i^{(k)}} + \epsilon} \ , \tag{3.40}$$

where ϵ is a very small value (*i.e.,* 10^{-8}) to improve numerical stability. A good value for the hyper-parameters are $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (**Kingma and Ba** 2015). Throughout this thesis, we use either SGD with momentum or Adam to optimize cost functions for deep neural network training.

3.1.2.3 Second-order Methods

Consider a class of optimization with quadratic cost function, which is the form arise in least squares (see more details in Section 3.1.3). The objective function is denoted

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x}) \quad \text{with} \quad \mathbf{r}(\mathbf{x}) : \mathbb{R}^k \mapsto \mathbb{R}^m \ , \tag{3.41}$$

where the vector-valued function $\mathbf{r}(\mathbf{x})$ is referred as residual. We further assume a non-trivial case, where the residual $\mathbf{r}$ is non-linear with respect to $\mathbf{x}$.

Newton and Gauss-Newton The gradient of Equation (3.41) can be easily computed to be

$$\nabla f(\mathbf{x}) = \left(\frac{\partial \mathbf{r}(\mathbf{x})}{\partial \mathbf{x}}\right)^T \mathbf{r}(\mathbf{x}) = \mathbf{J}_\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x}) \tag{3.42}$$

with $\mathbf{J}_\mathbf{r}(\mathbf{x})$ being the Jacobian of $\mathbf{r}(\mathbf{x})$ as defined in Equation (3.30). Taking the Jacobian for the gradient, the Hessian is computed by

$$\mathbf{H}_f(\mathbf{x}) = \nabla\left(\mathbf{J}_\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x})\right) = \mathbf{J}_\mathbf{r}(\mathbf{x})^T\mathbf{J}_\mathbf{r}(\mathbf{x}) + \mathbf{H}_\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x}) \ . \tag{3.43}$$

Insert Equation (3.42) and Equation (3.43) in Equation (3.33), yield the Newton optimization update. The Newton iteration presents rapid convergence when the cost function is approximately quadratic near its minimal, but when the condition is not satisfied, it does not obviously speed up the convergence. Additionally, computing Hessian can be expensive.

Now with the further assumption that the linearization exists for the residual, the Hessian $\mathbf{H}_\mathbf{r}$ is hence zero and Equation (3.43) is simplified. The resulting update equation is the Gauss-Newton optimization, described as

$$\mathbf{J}_\mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{J}_\mathbf{r}(\mathbf{x})\Delta\mathbf{x} = -\mathbf{J}_\mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{r}(\mathbf{x}) \,. \tag{3.44}$$

Gauss-Newton Equation (3.44) is known as the normal equation, and the value $\mathbf{J}_\mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{J}_\mathbf{r}(\mathbf{x})$ is often used as a good approximation for Hessian.

Levenberg-Marquardt Levenberg-Marquardt (LM) optimization uses the following update

$$\left(\mathbf{J}_\mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{J}_\mathbf{r}(\mathbf{x}) + \eta\mathbf{I}\right)\Delta\mathbf{x} = -\mathbf{J}_\mathbf{r}(\mathbf{x})^\mathsf{T}\mathbf{r}(\mathbf{x}) \,. \tag{3.45}$$

From the discussion in Section 3.1.2.2, we see that gradient descent is obtained by replacing the Hessian with an arbitrary step size $\mathbf{H} = \eta\mathbf{I}$. Therefore, the LM optimization is a combination of Gauss-Newton and gradient descent. Note that depending on the result of the current iteration, the value of η changes for the next iteration. If $\Delta\mathbf{x}$ leads to minimization, the value of η is divided by a factor (*e.g.*, 10) for the next iteration. Otherwise, the value of η is multiplied by a factor. In this thesis, we use LM for graph optimization.

3.1.3 Least Squares

Least squares (LS) optimization have several applications throughout this thesis. In general, the LS problem can be described as follows. Consider a model $\mathbf{f}(\mathbf{x}, \theta) \in \mathbb{R}^m$ with input $\mathbf{x}_i \in \mathbb{R}^p$ and model parameter $\theta \in \mathbb{R}^k$. With n observations $\mathbf{f} = \left(\mathbf{f}(\mathbf{x}_1, \theta), \mathbf{f}(\mathbf{x}_2, \theta), \cdots, \mathbf{f}(\mathbf{x}_n, \theta)\right)^\mathsf{T}$, and the corresponding ground-truth values $\hat{\mathbf{f}} = \left(\hat{\mathbf{f}}_1, \hat{\mathbf{f}}_2, \cdots, \hat{\mathbf{f}}_n\right)^\mathsf{T}$, find the parameter θ^* that minimizes the following objective function

$$E_{LS}(\theta) = \frac{1}{2}\sum_i^n \left\|\mathbf{f}(\mathbf{x}_i, \theta) - \hat{\mathbf{f}}_i\right\|^2 \,. \tag{3.46}$$

Define $\mathbf{r}_i(\theta) = \mathbf{f}(\mathbf{x}_i, \theta) - \hat{\mathbf{f}}_i$ and use the matrix notation, the LS problem takes the form

$$\theta^* = \arg\min_\theta E_{LS}(\theta) = \arg\min_\theta \frac{1}{2}\sum_i^n \mathbf{r}_i(\theta)^\mathsf{T}\mathbf{r}_i(\theta) \,. \tag{3.47}$$

Linear versus Nonlinear It can be seen Equation (3.47) takes the same quadratic form as Equation (3.41). Hence nonlinear LS can be linearized with Equation (3.32) and optimized using the aforementioned second-order iterative method Gauss-Newton or LM algorithm. With linear LS, there exists a close-form solution. Let $\mathbf{r}_i(\theta) = A_i\theta + \mathbf{b}_i$ with $A_i \in \mathbb{R}^{m\times k}$ and $\mathbf{b}_i = -\hat{\mathbf{f}}_i$, yield

$$\frac{\partial E_{LS}(\theta)}{\partial\theta} = \sum_i^n \frac{\partial \mathbf{r}_i}{\partial\theta}\mathbf{r}_i = \sum_i^n A_i^\mathsf{T}\left(A_i\theta + \mathbf{b}_i\right) = 0 \quad\Longrightarrow\quad \mathbf{A}^\mathsf{T}\mathbf{A}\theta = -\mathbf{A}^\mathsf{T}\mathbf{b} \,, \tag{3.48}$$

where $\mathbf{A} = \sum_i^n A_i$ and $\mathbf{b} = \sum_i^n \mathbf{b}_i$. If $\mathbf{A}^\mathsf{T}\mathbf{A}$ is invertible, the solution is obtained as $\theta^* = -(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}\mathbf{b}$. If $\mathbf{A}^\mathsf{T}\mathbf{A}$ is not full rank, Moore-Penrose pseudo inverse is required.

A Probabilistic Equivalence From a probabilistic view, LS describes the ML estimation of a zero-mean unit variance Gaussian distribution. To see this, assume $\mathbf{r}(\theta) \sim p\left(\mathbf{r}(\theta) \mid 0, \mathbf{\Sigma}\right)$, and seek θ^* that maximize the likelihood. Using Equation (3.4), this is formulated by

$$\theta^* = \arg\min_{\theta} \sum_{i}^{n} -\log p\left(\mathbf{r}_i(\theta) \mid 0, \mathbf{\Sigma}\right) = \arg\min_{\theta} \sum_{i}^{n} \mathbf{r}_i(\theta)^\mathsf{T} \mathbf{\Sigma}^{-1} \mathbf{r}_i(\theta) \,. \tag{3.49}$$

Set covariance $\mathbf{\Sigma} = \sigma^2 \mathbf{I}$, Equation (3.49) coincides with Equation (3.47) and it becomes apparent LS optimization yield the ML estimation of a zero-mean unit variance Gaussian.

For Gaussian with a more general covariance, we now derive the iterative update. For simplicity in notation, derivation is done for each individual term $\mathbf{r}_i(\theta)$, and the subscription is dropped. Assuming a general case with nonlinear LS, the residual $\mathbf{r}(\mathbf{x})$ is linearized with Equation (3.34). Set the derivative of Equation (3.49) to zero gives,

$$\frac{\partial}{\partial \theta} \mathbf{r}_{\text{lin}}(\theta^{(k+1)})^\mathsf{T} \mathbf{\Sigma}^{-1} \mathbf{r}_{\text{lin}}(\theta^{(k+1)}) = \mathbf{J}^\mathsf{T}(\theta^{(k)}) \mathbf{\Sigma}^{-1} \left(\mathbf{r}(\theta^{(k)}) + \mathbf{J}(\theta^{(k)}) \Delta\theta^{(k)} \right) = 0 \,, \tag{3.50}$$

Further Expand this result, and sum up all the residual terms, yield the following normal equation,

$$\sum_{i}^{n} \mathbf{J}_i(\theta^{(k)})^\mathsf{T} \mathbf{\Sigma}^{-1} \mathbf{J}_i(\theta^{(k)}) \Delta\theta = \sum_{i}^{n} -\mathbf{J}_i(\theta^{(k)})^\mathsf{T} \mathbf{\Sigma}^{-1} \mathbf{r}_i(\theta^{(k)}) \,. \tag{3.51}$$

To interpret this result, one way is that each dimension of the residual $\mathbf{r}$ is scaled accordingly with respect to the (co-)variance. For many multi-variables problems, $\mathbf{\Sigma}^{-1}$ estimate the optimal weights to combine the variables into the cost function, in comparison to some heuristic weighting factor. Recall that the ML estimation for Gaussian is derived with Equation (3.11).

Iteratively Re-weighted Least Squares Given every residual term contributes quadratically to the cost function, LS is known to be sensitive to outliers and data noise. Furthermore, we have seen that LS assumes Gaussian distribution, which is not always true in practical application. One way to reduce the problem is to replace the quadratic error term with some robust function $\rho(\cdot)$. Without a loss of generality, assume a real-valued residual $r(\mathbf{x}_i, \theta)$, and denote it $r_i(\theta)$ for simplicity. The revised robust optimization thus is formulated,

$$\theta^* = \arg\min_{\theta} E(\theta) = \arg\min_{\theta} \sum_{i}^{n} \rho\left(r_i(\theta)\right) \,. \tag{3.52}$$

Note that by setting ρ to be the ℓ_2 norm, Equation (3.52) falls back to the normal LS. To minimize $E(\theta)$, taking the derivative and setting it to zero, yield

$$\frac{\partial E(\theta)}{\partial \theta} = \sum_{i}^{n} \frac{\partial \rho(r)}{\partial r_i} \frac{\partial r_i(\theta)}{\partial \theta} = \sum_{i}^{n} \frac{\partial \rho(r)}{\partial r_i} \cdot \frac{1}{r_i(\theta)} \cdot r_i(\theta) \cdot \frac{\partial r_i(\theta)}{\partial \theta} \,, \tag{3.53}$$

The value $\psi(r_i) := \dfrac{\partial \rho(r)}{\partial r_i(\theta)}$ is called the influence function. Define $w\big(r_i(\theta)\big) := \dfrac{\psi(r_i)}{r_i(\theta)}$ to be the weight, the optimization defined in Equation (3.52) has an equivalent form

$$\theta^* = \arg\min_\theta E_{\text{IRLS}}(\theta) = \arg\min_\theta \sum_i^n w\big(r_i(\theta)\big)\left(\frac{1}{2} r_i^2(\theta)\right) . \tag{3.54}$$

Apparently E_{IRLS} has the form of weighted least squares, which has the same derivative with $E(\theta)$ in Equation (3.52) and hence share the same minimizer. Note that the weight $w(r_i(\theta))$ has an dependency on the residual, which changes every iteration. Therefore the robust LS is also known as the *iteratively re-weighted least squares* (IRLS).

With the definition of influence function and weight, it is easy to obtained that with normal LS defined in Equation (3.47), $\psi(r) = r$ and $w = 1$. This is as expected since all the samples contributed equally. For robust estimation, it is desirable that the influence function is even and decreases drastically when the absolute of residual increases. One commonly used robust function is the Huber norm $h_\delta(\cdot)$ with hyper-parameter δ, defined

$$h_\delta(r) = \begin{cases} \frac{1}{2} r^2 & \text{for} \quad |r| \leqslant \delta \\ \delta|r| - \frac{1}{2}\delta^2 & \text{for} \quad |r| > \delta \end{cases} , \tag{3.55}$$

which increases quadratically with small value of r, and linearly otherwise. The influence function and weight for Huber norm is

$$\psi(r) = \begin{cases} r & \text{for} \quad |r| \leqslant \delta \\ \delta \ \text{sign}(r) & \text{for} \quad |r| > \delta \end{cases} , \quad w(r) = \begin{cases} 1 & \text{for} \quad |r| \leqslant \delta \\ \dfrac{\delta}{|r|} & \text{for} \quad |r| > \delta \end{cases} . \tag{3.56}$$

Relation to M-estimator In literature, the robust error function ρ is also known as M-estimators as proposed by **Huber** (1972), where M stands for maximum-likelihood. As shown before, LS assumes Gaussian distribution, which amounts to using ℓ^2-norm for robust function. We now derive t-distribution leads to IRLS optimization. Using the result of Equation (3.49), and substitute $\mathbf{r}(\theta) \sim p(\mathbf{r}(\theta \mid 0, \boldsymbol{\Sigma}, \nu)$ instead of Gaussian, yield

$$\theta^* = \arg\min_\theta \sum_i^n \frac{\nu + k}{2} \log\left(1 + \frac{\mathbf{r}_i(\theta)^T \boldsymbol{\Sigma}^{-1} \mathbf{r}_i(\theta)}{\nu}\right) . \tag{3.57}$$

This is already in the form of robust error function. Applying linearization, and sorting out the math to obtain the minimizer at 0 gradient, yield

$$\sum_i^n \underbrace{\frac{\nu + k}{\nu + \mathbf{r}_i(\theta^{(k)})^T \boldsymbol{\Sigma}^{-1} \mathbf{r}(\theta^{(k)})}}_{w_i(\mathbf{r})} \cdot \mathbf{J}_i^T(\theta^{(k)}) \boldsymbol{\Sigma}^{-1} \left(\mathbf{r}(\theta^{(k)}) + \mathbf{J}_i(\theta^{(k)}) \Delta\theta\right) = 0 , \tag{3.58}$$

which gives the normal equation with by

$$\sum_i^n w_i(\theta) \mathbf{J}_i^T(\theta^{(k)}) \boldsymbol{\Sigma}^{-1} \mathbf{J}_i^T(\theta^{(k)}) \theta = -\sum_i^n w_i(\theta) \mathbf{J}_i^T(\theta^{(k)}) \boldsymbol{\Sigma}^{-1} \mathbf{r}_i(\theta^{(k)}) . \tag{3.59}$$

For detailed review of different robust estimators, we direct the readers to **Huber** (1972).

3.2 Fundamentals on RGB-D Mapping and Tracking

In this section we provide the fundamentals on RGB-D cameras, direct dense tracking methods and 3D mapping techniques.

Notations Following the notations in Section 3.1, we further denote a 3D point by $\mathbf{v} = (x, y, z)^\mathsf{T}$ and a 2D pixel by $\mathbf{x} = (u, v)^\mathsf{T}$. The homogeneous coordinate is denoted with $\sim$ above, *i.e.,* homogeneous 3D point $\widetilde{\mathbf{v}} = (x, y, z, 1)^\mathsf{T}$ and 2D pixel $\widetilde{\mathbf{x}} = (u, v, 1)^\mathsf{T}$. An image or 2D structured data is considered as a 2D continuous domain $\Omega \subset \mathbb{R}^2$. A color image $I(\mathbf{x})$ is a mapping $\Omega : \mathbb{R}^2 \to \mathbb{R}^3$, and a depth image $D(\mathbf{x})$ is a mapping $\Omega : \mathbb{R}^2 \to \mathbb{R}$.

3.2.1 Sensing with RGB-D Cameras

The RGB-D camera is a type of sensor that captures real-time, high-resolution videos with paired[1] color image and dense depth image. The availability of RGB-D cameras encouraged rapid progress in algorithms that use dense depth measurements for visual SLAM, robotic vision, dense 3D reconstruction, and augmented reality.

There are many popular commodity RGB-D cameras, *e.g., Microsoft Kinect V1*, *Kinect V2*, *Asus Xtion Pro* and *Intel RealSense*. All these sensors have an active depth camera that emits light, as opposed to passive cameras which only observe light. The techniques for depth measurement are slightly different. The Kinect V1, Xtion Pro and RealSense are all based *structure light*, where a known structured near-infrared pattern is projected into the scene with a laser projector, and observed by an infrared camera. The depth is obtained with stereo vision, where correspondences are searched between the projected and observed dotted patterns, then triangulation is performed for successful matches to estimate depth. In addition to occlusions caused by object shadows, depth measurements are often missing for dark, reflective or thin surfaces. The measurements are typically accurate up to a few meters and have near-field occlusion (invalid measurements within *e.g.,* 0.3 meter). Outdoor is another limitation, because the infrared light in sunlight overwhelms the infrared camera. Kinect V2 uses time of flight (ToF) technology for depth, which emits light pulses and measures the light travel time from the emission to receiving the reflection. Kinect V2 has a large depth range, but is also heavier and requires more power.

Kinect V1 and Asus Xtion Pro are the two major RGB-D cameras used in this thesis. They have very similar performance, where both provide synchronized 640×480 color video and 640×480 depth video at 30 Hz frame rate[2]. A detailed analysis of the depth noise is provided by **Khoshelham** (2011).

[1] A paired color and depth images are taken simultaneously in time. However, in practice, there is always a small time difference. This is difference however is ignored in this thesis.

[2] The intrinsic depth resolution is 320×240, but standard drivers produce upsampled resolution. With reduced resolution 320×240, the sensor can capture at 60 Hz. Unless otherwise stated, 640×480 is the resolution we work with.

Pinhole Camera Model The *pinhole camera model* approximates the projective mapping between 3D points and the corresponding 2D image coordinates. An ideal pinhole camera assumes the aperture is a point and no lens is used to focus light. The model does not include the geometric distortions, the blurring of unfocused objects, and the finite sized apertures. The model validity depends on the quality of the camera and in general, the model accuracy decreases from the center of the image to the edges as lens distortion increases.

The pinhole camera model consists of two set of parameters, the *intrinsics* and the *extrinsics*. The extrinsic parameters describe the pose of a camera given some global coordinate system (detailed in Section 3.2.2). The intrinsics describes the image formation with perspective projection. Usually, the intrinsic parameters are organized into a matrix $\mathbf{K}$ given by

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} , \tag{3.60}$$

where f_x and f_y are the focal length of the horizontal X and vertical Y direction in pixel units, respectively. The parameters c_x and c_y describe the principal point on the image plane, which is the center of the image providing no distortion. The parameter γ is the skew coefficient between the X and Y axis. Unless otherwise stated, γ is set 0 in this thesis. The process of obtaining the intrinsics is done with camera calibration.

Given a pinhole camera and the intrinsics, the perspective projection $\rho : \mathbb{R}^3 \mapsto \mathbb{R}^2$ is defined

$$\tilde{\mathbf{x}} = \rho(\mathbf{v}) = \frac{1}{z}\mathbf{K}\mathbf{v} , \qquad \Longrightarrow \begin{cases} u = x f_x / z + c_x \\ v = y f_y / z + c_y \end{cases} , \tag{3.61}$$

which obtains the continuous 2D pixel coordinates given a 3D point. Given the depth value d, the inverse $\rho^{-1} : \mathbb{R}^2 \mapsto \mathbb{R}^3$, known as the back-projection, is given by

$$\mathbf{v} = \rho^{-1}(\mathbf{x}) = d\mathbf{K}^{-1}\tilde{\mathbf{x}} \qquad \Longrightarrow \begin{cases} x = (u - c_x)d/f_x \\ y = (v - c_y)d/f_y \\ z = d \end{cases} . \tag{3.62}$$

3.2.2 Direct Methods for Motion Estimation

The extrinsics of pinhole camera describe how a local camera coordinate is transformed to align with the global coordinates. The calculation of the extrinsics is referred as camera tracking, motion estimation or pose estimation. Tracking with vision data, *i.e.,* using sequences of images, is also known as the problem of *visual odometry*.

3.2.2.1 Representations of Rigid Body Motion

This thesis only considers *rigid body motion*, which is composed of a 3D rotation and a 3D translation with in total six degrees of freedom (6 DoF). Rigid transformation preserves the distance between any pair of points as well as their orientations. The space of all rigid body motions forms a Lie group, known as the *special Euclidean group* $\mathbb{SE}(3)$. Hence rigid body motion is also called the special Euclidean transformation. Following we discuss the common representations for rigid body motion.

Transformation Matrix A common representation of the rigid body motion is the *transformation matrix* $\mathbf{M} \in \mathbb{R}^{4\times4}$, also known as the homogeneous representation,

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] , \tag{3.63}$$

where $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is the translation vector. In particular, the rotation matrix $\mathbf{R}$ belongs to the *special orthogonal group* $\mathbb{SO}(3)$, which preserves the orientation,

$$\mathbb{SO}(3) = \left\{ \mathbf{R} \in \mathbf{R}^{3\times3} \,\middle|\, \mathbf{R}^T\mathbf{R} = \mathbf{I} \,, \det(\mathbf{R}) = 1 \right\} . \tag{3.64}$$

The special Euclidean group $\mathbb{SE}(3)$ is therefore defined

$$\mathbb{SE}(3) = \left\{ \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} \,\middle|\, \mathbf{R} \in \mathbb{SO}(3) \,, \mathbf{t} \in \mathbb{R}^3 \right\} . \tag{3.65}$$

The inverse transformation can be easily computed to be

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ 0^T & 1 \end{bmatrix} . \tag{3.66}$$

To transform a 3D point, this is given

$$t(\mathbf{M}, \boldsymbol{v}) = \mathbf{R}\boldsymbol{v} + \mathbf{t} = \mathbf{M}\widetilde{\boldsymbol{v}} . \tag{3.67}$$

The transformation matrix provides a straight-forward computation to transform a point with matrix multiplication. With incremental transformation, it can be obtained easily with left matrix multiplication. The disadvantage is that the transformation matrix is an over-parameterization of the 6 DoF rigid body motion. This causes to complications in parameter optimization, where extra constraints must be introduced to ensure $\mathbf{R}$ is orthonormal.

Exponential Coordinates A minimum parameterization of the rigid body motion can be obtained with the canonical *exponential coordinates*, also known as the *twist coordinate*. The exponential coordinates ξ is defined

$$\xi = (\nu_1, \nu_2, \nu_3, \omega_1, \omega_2, \omega_3)^T = \left(\boldsymbol{\nu}^T, \boldsymbol{\omega}^T\right)^T \in \mathbb{R}^6 , \tag{3.68}$$

where $\mathbf{v} = (v_1, v_2, v_3)^\mathsf{T}$ is the linear velocity and $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^\mathsf{T}$ is the angular velocity. The exponential coordinates define the *twist* $\hat{\xi}$ by

$$\hat{\xi} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{v} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix} . \tag{3.69}$$

The operator $[\cdot]_\times$ generates a skew-symmetric matrix from a vector. The space of all skew-symmetric matrices forms the *tangent space* of the $\mathbb{SO}(3)$ group,

$$\mathfrak{so}(3) = \left\{ [\mathbf{a}]_\times \in \mathbb{R}^{3\times 3} \mid \mathbf{a} \in \mathbb{R}^3 \right\} . \tag{3.70}$$

The set of all exponential coordinates forms the tangent space, also known as *Lie algebra*, of $\mathbb{SE}(3)$ group

$$\mathfrak{se}(3) = \left\{ \hat{\xi} = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{v} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix} \middle| [\boldsymbol{\omega}]_\times \in \mathfrak{se}(3) , \mathbf{v} \in \mathbf{R} \right\} . \tag{3.71}$$

The conversion between the exponential coordinates and the transformation matrix is obtained by the *exponential map* $g : se(3) \mapsto \mathbb{SE}(3)$,

$$\mathbf{M} = g(\xi) = \exp\left(\hat{\xi}\right) = \mathbf{I} + \hat{\xi} + \frac{\hat{\xi}^2}{2!} + \ldots + \frac{\hat{\xi}^n}{n!} + \ldots . \tag{3.72}$$

For small motion, *i.e.*, $\xi \approx \mathbf{0}$, the exponential map can be approximated around the identity $\mathbf{M} \approx \mathbf{I} + \hat{\xi}$. Using the Rodriguez' formula, a closed-form solution can be obtained,

$$\mathbf{R} = \exp\left([\boldsymbol{\omega}]_\times\right) = \mathbf{I} + \frac{\sin\left(\|\boldsymbol{\omega}\|\right)}{\|\boldsymbol{\omega}\|}[\boldsymbol{\omega}]_\times + \frac{1-\cos\left(\|\boldsymbol{\omega}\|\right)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times^2 , \tag{3.73}$$

$$\mathbf{t} = \left(\mathbf{I} + \frac{1-\cos\left(\|\boldsymbol{\omega}\|\right)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times + \frac{\|\boldsymbol{\omega}\| - \sin\left(\|\boldsymbol{\omega}\|\right)}{\|\boldsymbol{\omega}\|^3}[\boldsymbol{\omega}]_\times^2\right)\mathbf{v} . \tag{3.74}$$

Denote the closed-form solution evaluated by ξ_k by $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$. The Jacobian of $g(\xi)$ with respect to ξ is calculated to be

$$\frac{\partial g(\xi)}{\partial \xi} = \begin{pmatrix} \mathbf{0} & -[\mathbf{r}_1]_\times \\ \mathbf{0} & -[\mathbf{r}_2]_\times \\ \mathbf{0} & -[\mathbf{r}_3]_\times \\ \mathbf{I} & -[\mathbf{t}]_\times \end{pmatrix} \in \mathbb{R}^{12\times 6} \text{ , with } \begin{aligned} \mathbf{r}_1 &= (r_{11}, r_{21}, r_{31})^\mathsf{T} \\ \mathbf{r}_2 &= (r_{12}, r_{22}, r_{32})^\mathsf{T} \\ \mathbf{r}_3 &= (r_{13}, r_{23}, r_{33})^\mathsf{T} \end{aligned} . \tag{3.75}$$

The inverse of the exponential map $g(\xi)$ is the *logarithm map* $g^{-1} : \mathbb{SE}(3) \mapsto \mathfrak{se}(3)$,

$$\xi = g^{-1}(\mathbf{M}) = \log(\mathbf{M}) . \tag{3.76}$$

The exponential coordinates are not defined in the Euclidean manifold. Therefore, the Euclidean vector addition cannot be used to add twists. We define the motion composition operator $\oplus$ with the left multiplication by,

$$\xi \oplus \Delta\xi = g^{-1}\big(g(\Delta\xi)g(\xi)\big) \ . \tag{3.77}$$

For simplicity, the inverse motion is denoted ξ^{-1}, defined by

$$\xi^{-1} = g^{-1}\left(g(\xi)^{-1}\right) \ . \tag{3.78}$$

The rigid body transformation with the twist representation is given by,

$$t(\xi, \mathbf{v}) = g(\xi)\mathbf{v} \ . \tag{3.79}$$

Let $\mathbf{v}_t \equiv t(\xi, \mathbf{v})$, using the result from Equation (3.75), the Jacobian of $t(\xi, \mathbf{v})$ with respect to ξ can be obtained by

$$\frac{\partial t(\xi, \mathbf{v})}{\partial \xi} = \frac{\partial t(\xi, \mathbf{v})}{\partial g} \cdot \frac{\partial g(\xi)}{\partial \xi} = \begin{pmatrix} \mathbf{I} & -[\mathbf{v}_t]_\times \end{pmatrix} \in \mathbb{R}^{3\times 6} \ . \tag{3.80}$$

Quaternion *Quaternion* is another alternative to represent rotations, which in combination with a translation vector $\mathbf{t} \in \mathbb{R}^3$ can be used to parameterize the rigid body motion. Quaternions are defined by a generalized complex number $q = q_0 + q_1 i + q_2 j + q_3 ij$, with $i^2 = -1$, $j^2 = -1$ and $ij = -ji$. The rotation group $\mathbb{SO}(3)$ can be fully represented by the set of all unit quaternions $\mathbf{q}$,

$$\mathbb{S}^3 = \left\{\mathbf{q} = (q_x, q_y, q_z, q_w)^T \mid \|\mathbf{q}\|^2 = q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1\right\} \ . \tag{3.81}$$

The unit quaternion $\mathbb{S}^3$ describes the unit sphere in $\mathbb{R}^4$. Unit quaternions provide a smooth representation without singularity. In comparison to exponential coordinates, unit quaternions are less ambiguous. One rotation is mapped to two unit quaternions, whereas to infinite many exponential coordinates. Additionally, quaternions are defined in Euclidean space. Therefore two quaternions can be directly added and then normalized to have unit length. The conversion to the rotation matrix is defined

$$\mathbf{R} = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_x q_w - 2q_w q_z & 2q_x q_z + 2q_y q_w \\ 2q_x q_y + 2q_z q_w & 1 - 2q_x^2 - 2q_z^2 & 2q_y q_z - 2q_x q_w \\ 2q_x q_z - 2q_y q_w & 2q_y q_z + 2q_x q_w & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix} \ . \tag{3.82}$$

3.2.2.2 Direct Camera Tracking

Consider two RGB-D images (I_1, D_1) and (I_2, D_2), the goal is to estimate the 6 DoF rigid body transformation from the second image to the first image, ξ_{12}.

Direct Tracking with Color: the Lukas-Kanade Image Alignment Algorithm One canonical solution to estimate the transformation between two color images is the Lukas-Kanade (LK) image alignment algorithm (**Baker and Matthews** 2004). Under the photo-consistency assumption, the error function of LK algorithm is formulated by

$$E_{LK}(\xi_{12}) = \sum_{x_i \in \Omega_2} r_{x_i}^2(\xi_{12}) = \sum_{x_i \in \Omega_2} \left(I_1\left(\omega(\xi_{12}, x_i)\right) - I_2(x_i)\right)^2 \quad , \tag{3.83}$$

where $x_i \in R^2$ is the i-th pixel in the second image. The function ω defines the *warping* from the second image I_2 to the first image I_1. Using Equation (3.61), Equation (3.62) and Equation (3.79), warping is defined

$$\omega(\xi, x_i) = \rho\Big(t\left(\xi, \rho^{-1}(x_i)\right)\Big) \; . \tag{3.84}$$

The warping function requires depth estimation for back-projection ρ^{-1}, which is directly available with RGB-D cameras. The major problem with the LK algorithm is the brightness consistency assumption, which only holds true for surfaces with Lambertian reflection. However, most natural materials are not Lambertian, where the brightness of surfaces changes given different view directions. Additionally, RGB-D cameras often have auto-exposure, and lighting conditions vary during capture. Note that the depth measurement from RGB-D cameras also contain noise. These noise source all contribute to outliers on a quadratic error formulation. To improve the robustness, **Kerl et al.** (2013b) proposed the robust LK formulation for RGB-D data with M-estimators.

Direct Tracking with Depth: the Iterative Closest Point Algorithm Consider two 3D shapes, represented by point clouds, the iterative closest point (ICP) algorithm (**Besl and McKay** 1992; **Y. Chen and Medioni** 1991) attempts to find the optimal transformation between the shapes such that the total alignment error is minimized. To measure the alignment error, the point-to-point distance is used in **Y. Chen and Medioni** (1991), and the point-to-plane distance is proposed in **Besl and McKay** (1992). Given a depth image, a dense point cloud can be obtained by back-projection. Therefore direct image alignment can be formulated with ICP.

Assume the surface normal is be estimated to be $N_1(x) : \mathbb{R}^2 \mapsto \mathbb{R}^3$. Further assume a set of correspondences can be obtained[3], where pixel $x \in \mathbb{R}^2$ in the second image is associated with pixel x' in the first image. The error function of point-to-plane ICP is formulated,

$$E_{ICP}(\xi_{12}) = \sum_{x_i \in \Omega_2} r_{x_i}^2(\xi_{12}) = \sum_{x_i \in \Omega_2} \Big(N_1(x_i')^T \left(t(\xi_{12}, \rho^{-1}(x_i)) - \rho^{-1}(x_i')\right)\Big)^2 \; . \tag{3.85}$$

Consider the noisy depth measurement from RGB-D cameras, there are several solutions to improve the robustness. **Newcombe et al.** (2011a) proposed to align the current depth

[3] One solution is to use the projective data association proposed in **Newcombe et al.** (2011a), which uses warping as defined in Equation (3.84) to initialize data association and only accepts correspondences when the points are geometrically consistent.

image to a de-noised depth, which can be considered as an average depth estimation. **Kerl et al.** (2013a) proposed to use robust least squares. **Gutierrez-Gomez et al.** (2016) proposed to use inverse depth, given that the disparity has Gaussian noise.

Jacobian Computation The error functions Equation (3.83) and Equation (3.85) are both nonlinear least squares, which can be solved iteratively with Gauss-Newton. We now derive the Jacobian for the residual. For simplicity, the subscript is dropped. Linearize the residual with Equation (3.34) at the last iteration $\xi_{12}^{(k)}$, apply the chain rule to calculate the gradient and use the result of Equation (3.80), the gradient of LK and ICP both take the form,

$$J = \left(a^T \quad (v_t \times a)^T\right)\Big|_{\xi_{12}^{(k)}} \in R^6 \quad , \tag{3.86}$$

where $v_t = t\,(\xi, \rho(x)) = (x_t, y_t, z_t)^T$ evaluated at $\xi_{12}^{(k)}$. With the LK algorithm, the term a_i is computed to be

$$a_{LK} = \left(\frac{g_x f_x}{z_t}, \frac{g_y f_y}{z_t}, -\frac{g_x f_x + g_y f_y}{z_t^2}\right)^T \quad , \tag{3.87}$$

where g_x, g_y are the image gradient, f_x, f_y are the focal length. For the ICP algorithm,

$$a_{ICP} = N_1(x') = (n'_x, n'_y, n'_z)^T \quad . \tag{3.88}$$

Summing up the Jacobian with respect to all the correspondences, and insert into Equation (3.44), the incremental update $\Delta\xi$ can be obtained by solving the normal equation. The composition of motion is then performed in the tangent space $\xi^{(k+1)} = \Delta\xi \oplus \xi^{(k)}$ as defined in Equation (3.77).

Consider visual and geometrical measurements are often complementary to each other, it is desirable to combine the LK and ICP algorithm for RGB-D tracking. Such methods proposed by **Gutierrez-Gomez et al.** (2016); **Kerl et al.** (2013a); **Whelan et al.** (2015b) have shown to improve tracking accuracy. In this thesis, we develop our tracking algorithm based on both the LK algorithm and the ICP algorithm.

3.2.3 3D Geometry

3.2.3.1 Map Representation

There are many ways to represent 3D maps, *e.g.*, point clouds, surfels, meshes, range maps, depth images, occupancy map, signed distance function (SDF) and others. This thesis mainly uses point clouds, meshes and SDFs embedded in volumetric representation.

Point Cloud, Mesh A point cloud provides one simple map representation in the form of 3D points $\{\mathbf{p}_i \mid \mathbf{p}_i(x_i, y_i, z_i) \in \mathbb{R}^3\}$. Optionally, each point can have other attributes, *e.g.*, color value $\mathbf{c} \in \mathbb{R}^3$, surface normal $\mathbf{n} \in \mathbb{R}^3$, curvature $\zeta \in \mathbb{R}$. Usually point clouds are unorganized. The KdTree provides an efficient solution to find the neighboring points. With synchronized RGB-D images, it is straight-forward to obtain an organized colored point cloud maintaining the same neighborhood as images. However, it is also important to note two points adjacent in the 2D image are not necessarily close in 3D.

Point clouds are good for visualization, but they do not provide useful a description about the surfaces and topology. Points do not have size, therefore they cannot infer occlusion either. The alternative is to use meshes, where points (referred as vertices) are connected into polygons (referred as faces) to describe the surfaces. Usually meshes do not mix different polygon types, and this thesis only considers triangular meshes or quad meshes. Meshes can be considered as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where vertices $\mathcal{V}$ are connected by edges $\mathcal{E}$. Using this connectivity, neighborhood searches over the mesh manifold can be done along the edges.

Volumetric Signed Distance Function Given a set Ω and a point x, the *signed distance function* (SDF) is a level set that yields the distance of a point to the boundary of the Ω. Usually the value is positive for points inside Ω and negative for points output Ω. Proposed in **Curless and Levoy** (1996), a 3D volumetric presentation is used to store the SDFs, where surfaces are embedded into the zero *isosurface*. Voxels in front of surfaces get positive SDF values, and negative otherwise. To integrate multiple measurements into a de-noised complete model, a weighted fusion for the SDF can be applied. Given a 3D SDF volume denoted by $S(\mathbf{x}) : \mathbb{R}^3 \mapsto \mathbb{R}$, and the associated weight $W(\mathbf{x}) : \mathbb{R}^3 \mapsto \mathbb{R}$, the individual SDF measurement is given by $d_i(\mathbf{x})$, fusion as proposed in **Curless and Levoy** (1996) is defined

$$S(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x}) d_i(\mathbf{x})}{\sum_i w_i(\mathbf{x})} \quad , \quad W(\mathbf{x}) = \sum_i w_i(\mathbf{x}) \quad . \tag{3.89}$$

If weight $w_i(\mathbf{x})$ is set to uniform, this fusion essentially assumes Gaussian noise in the measurements. Equation (3.89) is equivalent to the following incremental update

$$S_{i+1}(\mathbf{x}) = \frac{W_i(\mathbf{x}) S_i(\mathbf{x}) + w_{i+1}(\mathbf{x}) d_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})} \quad , \quad W_{i+1}(\mathbf{x}) = W_i(\mathbf{x}) + w_{i+1}(\mathbf{x}) \quad . \tag{3.90}$$

The SDF represents surfaces implicitly as level sets. To extract meshes embedded in the zero-isosurface, the marching cubes algorithm by **Lorensen and Cline** (1987) is usually applied.

The volumetric SDF representation is shown useful by **Newcombe et al.** (2011a) to integrate multi-view depth images from Kinect into consistent 3D reconstructions. Further **Newcombe et al.** (2011a) proposed to truncate the SDF for efficiency and to saturate $W(\mathbf{x})$ to a predefined threshold to give priority for newer measurements. **Whelan et al.** (2015b) proposed to fuse the color images similar to depth fusion, and consequently reconstruct colored meshes. In this thesis, we use SDF volumes for RGB-D fusion and investigate different weighting strategies as detailed in Chapter 10.

3.2.3.2 Principal Component Analysis in $\mathbb{R}^3$

Principal component analysis (PCA) performs the orthogonal projection of data onto a lower dimensional principal subspace, such that the variance of the projection is maximized. For

3D data, PCA can be used to estimate important local properties, *e.g.*, surface normal and curvature. Given a set of k observations $\{\mathbf{x}_i\}$ with the data mean and variance

$$\boldsymbol{\mu}_x = \frac{1}{k}\sum_i^k \mathbf{x}_i \quad , \quad \boldsymbol{\Sigma}_x = \frac{1}{k}\sum_i^k (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 \end{bmatrix} \quad . \tag{3.91}$$

With covariance $\boldsymbol{\Sigma}_x$ being positive semi-definite, the principal values and principal vectors coincide with the eigenvalues $\lambda_1, \lambda_2, \lambda_3$ and eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ of $\boldsymbol{\Sigma}_x$. Unless otherwise stated, this thesis assumes $\lambda_1 \leqslant \lambda_2 \leqslant \lambda_3$.

Surface Normal, Curvature A common estimate for surface normal and curvature is based on PCA (**Mitra et al.** 2004; **Pauly et al.** 2002). Given a 3D point, PCA can be performed given k nearest neighbors (kNN). The surface normal can be approximated by $\mathbf{u}_1$ and the surface curvature can be approximated with

$$\zeta = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \quad . \tag{3.92}$$

3.2.3.3 Plane Fitting

There are many ways to describe a plane, *e.g.*, with three non-colinear points, two distinct but intersecting lines or two parallel lines. Each of these method contributes a way for parameterization. In this thesis, a plane is characterized by the Hessian normal form $\boldsymbol{\pi} = (\mathbf{n}^T, d)^T \in \mathbb{R}^4$. With this parameterization, the unit vector $\mathbf{n}$ points to the normal of the plane, and $\mathbf{n}^T\mathbf{x} + d$ measures the distance of a point $\mathbf{x}$ to the plane.

Fitting Plane to Points Consider $k, k \geqslant 3$ number of 3D points $\mathbf{x}_i$, the estimation of $\boldsymbol{\pi}$ can be formulated into a LS minimization

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} E(\boldsymbol{\pi}) = \arg\min_{\boldsymbol{\pi}} \frac{1}{k}\sum_i^k \left(\mathbf{n}^T\mathbf{x}_i + d\right)^2 \quad . \tag{3.93}$$

Setting the derivative with respect to d to zero, yields to following

$$\frac{1}{k}\sum_i^k \mathbf{n}\mathbf{x}_i + d = 0 \quad \Longrightarrow \quad d = \mathbf{n}^T\frac{1}{k}\sum_i^k \mathbf{x}_i = \mathbf{n}^T\boldsymbol{\mu}_x \quad . \tag{3.94}$$

The mean $\boldsymbol{\mu}_x$ is also known as the centroid. Insert Equation (3.94) back into Equation (3.93) to optimize with respect to $\mathbf{n}$ and use the covariance defined in Equation (3.91), yield

$$\mathbf{n}^* = \arg\min_{\mathbf{n}} \frac{1}{k}\sum_i^k \mathbf{n}^T(\mathbf{x}_i - \boldsymbol{\mu}_x)(\mathbf{x}_i - \boldsymbol{\mu}_x)^T\mathbf{n} = \arg\min_{\mathbf{n}} \mathbf{n}^T\boldsymbol{\Sigma}_x\mathbf{n} \quad , \tag{3.95}$$

The eigenvector $\mathbf{u}_i$ corresponding to the smallest eigenvalue of $\boldsymbol{\Sigma}_x$ is the minimizer to Equation (3.95). With $\mathbf{n}$ solved, d becomes apparent with Equation (3.94). We remark the residual of plane fitting is given by λ_1.

Incremental Plane Merging LS plane fitting requires to evaluate the mean and covariance of the points, and performs the eigenvalue decomposition on the covariance. Consider two clusters of points of size n_1 and n_2, where LS is performed separately to estimate plane, the merged plane estimation can then be computed using the existing calculation of $\mu_1, \mu_2, \mathbf{\Sigma}_1, \mathbf{\Sigma}_2$. Denote the merged centroid and covariance with $\mu_{12}, \mathbf{\Sigma}_{12}$,

$$\mu_{12} = \frac{n_1\mu_1 + n_2\mu_2}{n_1 + n_2} \quad ; \quad \mathbf{\Sigma}_{12} = \frac{n_1\mathbf{\Sigma}_1 + n_2\mathbf{\Sigma}_2}{n_1 + n_2} - (n_1\mathbf{\Sigma}_1) \circ (n_2\mathbf{\Sigma}_2) \ , \tag{3.96}$$

with the operator $\circ$ denoting the element-wise Hadamard product. To see this result, we make use of the equality $\sigma_{xy}^2 = \mathbb{E}((x - \mu_x)(y - \mu_y)) = \mathbb{E}(xy) - \mathbb{E}(x)\mathbb{E}(y)$ for covariance estimation. The incremental plane merging provides an efficient solution to detect planar surfaces with batch processing. We use this method to detect planes given organized point cloud obtained from depth images (detailed in Chapter 5).

3.3 Convolutional Neural Networks in A Nutshell

Deep *convolutional neural networks* (CNNs) have been shown to be a powerful model for many machine learning and computer vision problems. The term *neural network* originates from the attempt to find a mathematical representation that models biological neural systems with respect to how information is processed. The term *convolutional* points out the two distinctions of the CNNs to the other types of *artificial neural networks* (ANNs), local fully connection with weight sharing and feed-forward inference with error back-propagation. Last the word *deep* emphasizes that the networks consist of a large number of hidden layers as opposed to traditional ANNs with a few fully-connected layers.

Notations Consider a mini batch data $\mathcal{B}$. We denote the input and output of a layer by $\mathbf{X} = \{\mathbf{x}\}_{\mathcal{B}}$ and $\mathbf{Y} = \{\mathbf{y}\}_{\mathcal{B}}$. Both $\mathbf{X}$ and $\mathbf{Y}$ are 4D tensors of the shape $B \times K \times H \times W$, with batch size B, channel number K, spatial height H and spatial width W. Except the batch size, all other three dimensions may differ between the input and the output. Without loss of generality, we drop the batch notation and denote $\mathbf{x} = \left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(K)}\right)$, and use x_{ij}^k to refer the spatial location of channel k. For simplicity we occasionally drop the indices and use x without causing confusion.

Model of Single Neurons The core component of a neural network is how biological neurons are represented by a mathematical model. Consider a simplified biological model, a neuron collects the input signals through multiple *dendrites*, processes them inside the cell and outputs the signals along a single *axon* in the form of biological impulse. The end of axon are *terminals*, which are connected to other neurons via *synapses* for message passing. To model this process, the mathematical model is defined as

$$y = f\left(\sum_{i}^{k} w_i x_i + w_0\right) \quad . \tag{3.97}$$

The term $\sum_i^k w_i x_i + w_0$ imitates the input, where the signal received from each dendrite as x_i through synapse as w_i are linearly combined. The function $f(\cdot)$ referred as activation, produces nonlinearity to resemble the output impulse.

From the canonical machine learning perspective, the model of a single neuron takes a similar form as the linear regression and classification models, which is defined

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_i^k w_i \phi_i(\boldsymbol{x}) + w_0\right) . \tag{3.98}$$

The function $\phi_i(\boldsymbol{x})$ is a set of fixed nonlinear basis functions, which is linearly combined with parameter $\boldsymbol{w}$ and bias with the parameter b. The activation function $f(\cdot)$ is nonlinear for classification and identity mapping for regression (**Bishop** 2006). With neural networks, the basis functions are represented by neurons. Instead of defining the basis functions, ANNs use learning to find the best configurations.

Feed-forward Inference and Error Back-propagation One important property of CNNs is the *feed-forward* inference, which means the output of layer ℓ only depends on the preceding layers, and not on any succeeding layers. Assume for a given layer ℓ, the mapping from the input to the output takes the form $\boldsymbol{y}^{(\ell)} = g\left(\boldsymbol{x}^{(\ell)}, \boldsymbol{w}^{(\ell)}\right)$, where $\boldsymbol{w}^{(\ell)}$ denotes the layer parameters if any and note $\boldsymbol{x}^{(\ell)} = \boldsymbol{y}^{(\ell-1)}$, the inference of CNN can be described

$$\boldsymbol{y}^{(\ell)}(\boldsymbol{x}, \boldsymbol{w}) = g^{(\ell)}\left(\boldsymbol{x}^{(\ell)}, \boldsymbol{w}^{(\ell)}\right) \tag{3.99}$$

$$= g^{(\ell)}\left(g^{(\ell-1)}\left(\cdots g^{(2)}\left(g^{(1)}\left(\boldsymbol{x}^{(1)}, \boldsymbol{w}^{(1)}\right), \boldsymbol{w}^{(2)}\right), \cdots, \boldsymbol{w}^{(\ell-1)}\right), \boldsymbol{w}^{(\ell)}\right) . \tag{3.100}$$

This expresses CNNs as a composition of functions. Consider a CNN with cost $E(\boldsymbol{x}, \boldsymbol{w})$ and total number of layers L. The gradient of the layer parameter $\boldsymbol{w}^{(\ell)}$ can be obtained by the chain rule. Using Equation (3.100), yields

$$\frac{\partial E(\boldsymbol{x}, \boldsymbol{w})}{\partial \boldsymbol{w}^{(\ell)}} = \frac{\partial E(\boldsymbol{x}, \boldsymbol{w})}{\partial g^{(L)}} \frac{\partial g^{(L)}(\boldsymbol{x}^{(L)}, \boldsymbol{w}^{(L)})}{\partial g^{(L-1)}} \cdots \frac{\partial g^{\ell+1}(\boldsymbol{x}^{(\ell+1)}, \boldsymbol{w}^{(\ell+1)})}{\partial g^{\ell}} \frac{\partial g^{\ell}(\boldsymbol{x}^{(\ell)}, \boldsymbol{w}^{(\ell)})}{\partial \boldsymbol{w}^{\ell}} , \tag{3.101}$$

which shows the gradient at layer ℓ can be composed from the gradient of all succeeding layers. More specifically, two gradients are evaluated by every layer evaluates at training. First, the gradient with respect to the current parameters $\nabla g^{(\ell)}_{\boldsymbol{w}^{(\ell)}}$ is computed and multiplied with gradients accumulated from all succeeding layers, so as to update $\boldsymbol{w}^{(\ell)}$. Second, the gradient with respect to the current input $\nabla g^{(\ell)}_{\boldsymbol{x}^{(\ell)}}$ is calculated and propagated further to the preceding layers. This parameter update scheme is referred as error *back-propagation* **LeCun et al.** (1989); **Rumelhart et al.** (1986).

Comparison to Other Neural Networks The regular ANNs is usually fully-connected. This means each neuron in the previous hidden layer is connected to every other neuron in the next layer. ANNs are known as universal approximators (**Bishop** 2006; **Stinchcombe**

and White 1989). A network with one hidden layer can uniformly approximate any continuous function, provided sufficient amount of hidden units. CNNs are a class of ANNs, hence they are also universal approximators. But instead of increasing the amount of neurons within a single layer, CNNs gain the approximation power from stacks of convolutional layers and increasing receptive field with pooling layers. As opposed to regular ANNs, CNNs require data being structured into a 4D tensor of the shape $B \times K \times H \times W$[4], and maintain such structure across all layers in order to perform convolution. Another important property of CNNs is the feed-forward architecture, which enables training with error back-propagation. Compared to the *recurrent neural networks* (RNNs) (**Graves et al.** 2009) and the *long short-term memory* networks (LSTMs)[5] by **Hochreiter and Schmidhuber** (1997), CNNs contain no memory of the past information, and are less suitable for learning from sequential data.

3.3.1 Basic Layers

3.3.1.1 Convolution

Convolutional layers are the core of CNNs. One convolutional layer is a collection of trainable linear filters (also known as kernels), which have small spatial resolution but fully extend along the input channel dimension. To perform filtering, each kernel convolves the input volume along the spatial dimension and produces one channel of the output volume. Given a C-channel input, apply a convolutional kernel with $F \times F$ spatial resolution at input location (i, j) yields,

$$\sum_{k}^{C} \sum_{h}^{F} \sum_{l}^{F} w_{l,h}^{k} x_{h+i-F,l+j-F}^{k} + b \quad , \tag{3.102}$$

where $x_{i,j}^{k}$ denotes the input neurons, $w_{l,h}^{k}$ denotes the weight parameter, and b is the bias parameter. The typical spatial resolution for convolutional filters is 3×3 and it hardly gets larger than 11×11. However, the number of filters contained by a convolutional layer varies from less than a hundred to a few thousands as the network gets deeper (typically doubles as the spatial resolution halves). This design of filtering, *i.e.,* spatially convolving small but deep filters over a volume of neurons is referred as *local fully connected with parameter sharing*. It is especially suitable for filtering images, which are naturally organized and locally correlated. The local fully connected layers are also essential to learn spatial invariant features. Due to the convolving operation, CNNs require structured data, and it is important that every hidden layer produces structured outputs where the spatial ordering is strictly maintained.

The output neuron volume of a convolutional layer is determined by four parameters: kernel size F, kernel numbers C and the input stride S and input padding P. Given an input of the shape $B_i \times C_i \times H_i \times W_i$, the output shape after convolutional layer is $B_i \times C \times H_o \times W_o$,

[4] The order of the dimension can be different given the implementation.
[5] LSTMs are a special class of RNNs.

with

$$H_o = \lfloor (H_i - F + 2P)/S \rfloor + 1 \quad ; \quad W_o = \lfloor (W_i - F + 2P)/S \rfloor + 1 \ . \tag{3.103}$$

1×1 Convolution Proposed in **M. Lin et al.** (2013), 1×1 convolution is a special configuration of convolution layers. As the convolution operation extends the full channel dimension, a 1×1 convolutional layers with C_o filters projects an input with C_i channels to an output with C_o channels with a linear transformation. The 1×1 convolution is therefore often used to adjust the volume depth, *e.g.*, for the bottleneck unit of ResNet by **K. He et al.** (2016a).

Dilated Convolution The dilated convolution (also known as atrous convolution) is proposed concurrently by **Yu and Koltun** (2016) and **L.-C. Chen et al.** (2015). The motivation for dilated convolution is to increase the receptive field of a kernel without increasing the amount of parameters. For this purpose, the input is sampled with a stride to convolve with a kernel. Assume the input stride S_i, a kernel of size F effectively increases to $(F-1) \times (S_i - 1) + F$. Notice that a normal convolution have input stride $S_i = 1$. The dilated convolution is shown to improve the network performance for both classification (**Yu and Koltun** 2016) and segmentation (**L.-C. Chen et al.** 2015). However, the memory consumption of dilated CNNs is significantly higher, which is undesirable for training with large batch size. Therefore, in this thesis we do not use dilated convolution.

Fully-Connected Layers Fully-connected (FC) layers are the major layer type for canonical ANNs and commonly used as the last few layers in CNN classifiers. FC layers typically requires a lot of parameters, given every single neuron of the previous layer is connected to every single neuron in the current layer with a different weight. FC layers therefore have the receptive field of the entire input. Suggested in **Long et al.** (2015), FC layers can be converted by convolutional layers, *e.g.*, a FC layer over 7×7 spatial input is equivalent to a convolutional layer with 7×7 kernel size, convolved with stride 7 and zero padding. Replacing FC with convolutional layers also has the advantage that the same network can be used with different input image size. In this thesis, we use construct fully convolutional CNNs.

Transposed Convolution Many works in the literature, often misuse deconvolution to mean transposed convolution. In this thesis, we distinguish these two concepts. We refer deconvolution strictly as the mathematical operation to reverse the effect of convolution on data [6]. Transposed convolution, also known as fractional convolution, sub-pixel convolution or upconvolution, is a layer that combines upscaling and convolution into one single operation. Transposed convolution was first introduced in **M. D. Zeiler et al.** (2010), further developed in **M. D. Zeiler et al.** (2011). It became a popular technique for upsampling after **M. D. Zeiler and Fergus** (2014) used it to visualize network features and **Long et al.**

[6] Consider a model $f \star g = h$, where f is the true signal, g is the noise and h is the observed signal, deconvolution attempts to recover f from observation h.

(2015) applied it to produce image segmentations. Conceptually, transposed convolution is equivalent to inserting zeros in-between two consecutive neurons and then convolve with a regular kernel.

3.3.1.2 Activation

Activation layers are element-wise operations to resemble the spikes of biological neurons to transmit signals. Convolution is a purely linear operation, to introduce nonlinearity CNNs rely on activation layers. These layers acts like switches that direct different information to be filtered through different paths. The common activation layers are listed below.

Sigmoid and Tanh The sigmoid activation is defined

$$f(x) \equiv \sigma(x) = \frac{1}{1+\exp(-x)} \ . \tag{3.104}$$

The sigmoid function takes in a real-valued scalar, and squashes it to the interval $(0, 1)$ with a monotonically increasing mapping. Sigmoid nicely resembles how biological neurons send out signal given the strength of input, with a continuous rate. However, sigmoid suffers two drawbacks that make it undesirable in deep CNNs. When the activation is close to either zero or one, the derivative saturates to zero, which kills all the signals from back-propagation and stops CNNs from learning. This problem is referred as *vanishing gradient*. Furthermore, sigmoid is not centered on zero, which always produces positive signals to the later layers. This may lead to undesirable dynamics in optimization, *e.g.*, zig-zagging patterns. A slightly preferred activation over sigmoid is the tanh function,

$$f(x) \equiv \tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)} = 2\sigma(2x)-1 \quad . \tag{3.105}$$

which linearly transforms $\sigma(x)$ to be zero-centered. However $\tanh(x)$ still suffers from the vanishing gradient problem. In this thesis, we do not use sigmoid or tanh for activation. However, sigmoid is also the logistic regression function, so they are often used in output layer for binary classification.

Rectified Linear Unit (ReLU), Leaky ReLU, Parameterized ReLU (pReLU) Proposed by **Krizhevsky et al.** (2012) for ImageNet classification, the rectified linear unit (ReLU) is one of most commonly used activation function. ReLU is defined

$$f(x) \equiv \mathrm{ReLU}(x) = \max(0, x) \quad , \tag{3.106}$$

which is simply a thresholding over zero. As shown in **Krizhevsky et al.** (2012), ReLU significantly accelerate the convergence in comparison to sigmoid/tanh. ReLU does not suffer from gradient vanish, but it can cause neurons to die during training. Due to the linear mapping, large gradients can be back-propagated via ReLU and eventually lead parameters to be updated in the direction that some neurons become irreversibly inactivate. One attempt to fix this problem is to use the leaky ReLU defined by

$$f(x) \equiv \text{pReLU}(x) = \max(0, x) + \alpha \min(0, x) \quad , \tag{3.107}$$

where the hyper-parameter α is a very small constant (*e.g.*, 0.01) to enable gradient flow through negative activation. **K. He et al.** (2015) proposed to learn the hyper-parameter α and hence the naming parameterized ReLU. However, pReLU does not leads to consistent improvement as reported in literature.

Exponential Linear Unit (ELU) More recently proposed, the exponential linear unit(ELU) by **Clevert et al.** (2015) is computed by

$$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if} \quad x \leqslant 0 \end{cases} \quad , \tag{3.108}$$

with hyper-parameter α being a small constant to control when the negative activation saturates. As with ReLU, ELU also diminishes the vanishing gradient problem. Experiments also show ELUs help CNNs to converge faster and to better local minima.

3.3.1.3 Pooling and Unpooling

Pooling Pooling is a channel-wise operation that is important to reduce the space resolution and increase the receptive field. Typically pooling is performed by convolving a 2×2 window with stride 2 over each channel slice and extracting the maximum or average value for each filter location. This effectively reduces the spatial resolution by a factor of 2. Recently **Springenberg et al.** (2015) proposed to replace pooling with convolution. For simplicity, all the CNNs in this thesis use max pooling.

Unpooling Unpooling layers aim to increase the spatial resolution, which is important for CNNs to perform dense predictions. In **Long et al.** (2015) upsampling is done with transposed convolution, which requires cropping to achieve the desirable output resolution and therefore the receptive field is not well-aligned. An alternative is to use memorized unpooling proposed by **M. D. Zeiler et al.** (2011). This method works with paired pooling and unpooling layers, where the exact pooling locations are cached, and later used at unpooling to restore upsampling from low resolution input accordingly. The unpooled layers therefore are sparse, and further filtering is required to densify the output. In this thesis, we experiment with memorized unpooling in Chapter 6 and Chapter 7, and further explore other unpooling techniques in Chapter 8.

3.3.1.4 Auxiliary Layers

Dropout The dropout layer (**Srivastava et al.** 2014; **Wager et al.** 2013) introduces important regularization to CNNs to prevent over-fitting at training. As the name suggests, the dropout layer randomly turns off some neuron activations only at training. The dropout can be considered to sample a full network and only update parameters given the current sampling. Since no dropout is performed at testing, it can be interpreted as an averaging of a set of subnetworks. Implementation-wise dropout is achieved by keeping a neuron active with probability p, or setting it to zero otherwise. With this scheme, at training the expected neuron output is px. To maintain the same expectation at testing, the layer output needs to be scaled by p. To avoid scaling at test, a better solution is to scale active neurons by $1/p$ at training. This thesis use a default dropout rate $p = 0.5$.

Batch Normalization Batch normalization (also known as BatchNorm) is proposed by **Ioffe and Szegedy** (2015), which has become a common practice in neural networks, in particular for very deep networks. This layer is developed to reduce the problem of *internal covariance shift* in network training. This problem refers to the fact that, as the parameters of previous layers change during training, the input distribution of all the following layer also varies. This phenomenon cause difficulties in training, which requires lower training rate and careful initialization for the network to converge to a good local minimal. To address the problem, **Ioffe and Szegedy** (2015) proposed to enforce signal whitening before convolution layers. Given a mini batch $\mathcal{B}$, each input channel is normalized independently across the batch. The output is hence defined by

$$y_{ij}^{(k)} = \gamma^{(k)} \frac{x_{ij}^{(k)} - \mu_{\mathcal{B}}^{(k)}}{\sqrt{{\sigma_{\mathcal{B}}^{(k)}}^2 + \epsilon}} + \beta^{(k)} \tag{3.109}$$

where the channel-wise batch mean μ and variance σ^2 are calculated by

$$\mu_{\mathcal{B}}^{(k)} = \frac{1}{B}\sum_{n}^{B} x_n^{(k)} \quad , \quad {\sigma_{\mathcal{B}}^{(k)}}^2 = \frac{1}{B}\sum_{n}^{B} \left(x_n^{(k)} - \mu_{\mathcal{B}}^{(k)}\right)^2 \quad . \tag{3.110}$$

The variables γ and β are the scaling and shifting parameters to be learned from training. The parameter ϵ is a small constant to improve numerical stability. Setting $\gamma = 1, \beta = 0$ and ignoring ϵ, the batch normalization whitens the input signal to zero mean and unit variance. Batch normalization therefore preforms pre-processing on the layer input. With a non-trivial γ, β, the network learns the optimal linear transformation of the whitened signal. At inference time, γ and β are fixed to the global mean and variance recomputed given all the training data and the learned parameters. A common practice however is to aggregate γ, β with a moving average during training. In this thesis, we always apply batch normalization.

Shortcut Shortcut, also known as skip connection, is not a layer but a direct connection from a layer output to one of its succeeding layer, where at least one intermediate layer is skipped. Shortcut introduces identity mapping to CNNs, which is shown to be crucial for error back-propagation in very deep CNNs by **K. He et al.** (2016a). The identity mapping with shortcut is also useful to improve dense pixelwise predictions for encoder-decoder CNNs, where early-stage features from encoder are directly combined in the decoder (**Long et al.** 2015; **Ronneberger et al.** 2015).

3.3.1.5 Loss Layers

Cross-Entropy Loss for Multi-class Classification The cross-entropy loss for k-class classification is defined

$$E(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i}^{k} -q_i \log p_i(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i}^{k} -[\![i = c_g]\!] \log p_i(\boldsymbol{x}, \boldsymbol{w}) \quad , \tag{3.111}$$

where q_i denotes the true class probability and p_i is the predicted class probability. The cross-entropy loss then minimizes the KL divergence as defined in Equation (3.28). Further assume that the class distribution is one-hot with the true label c_g, yields the equivalent defnition as in Equation (3.111). To obtain the class probability, a popular choice is the softmax function, defined

$$\sigma_i(\boldsymbol{x}, \boldsymbol{w}) = \frac{\exp\left(g_i(\boldsymbol{x}, \boldsymbol{w})\right)}{\sum_j^k \exp\left(g_j(\boldsymbol{x}, \boldsymbol{w})\right)} \quad , \tag{3.112}$$

where $g_i(\boldsymbol{x}, \boldsymbol{w})$ is the classification score produced by the output layer (*e.g.*, with convolutional layer). The softmax function has a simple derivative, which is further simplified in combination with cross-entropy loss (also more stable numerically). The derivative is calculated to be

$$\frac{\partial \sigma_i(\boldsymbol{x}, \boldsymbol{w})}{\partial g_j} = \begin{cases} \sigma_i - \sigma_i \sigma_j & \text{if } i = j \\ -\sigma_i \sigma_j & \text{if } i \neq j \end{cases} \implies \frac{\partial - \log \sigma_i(\boldsymbol{x}, \boldsymbol{w})}{\partial g_j} = \begin{cases} \sigma_j - 1 & \text{if } i = j \\ \sigma_j & \text{if } i \neq j \end{cases} \quad . \tag{3.113}$$

The two-class softmax is the logistic regression function defined in Equation (3.104). There exist other loss functions for classification, *e.g.*, the hinge loss[7] which is often used in support vector machines. This thesis only uses the cross-entropy loss given its simple gradient computation and comparable performance with hinge loss. For semantic image segmentation, the loss is obtained by the summation over pixelwise cross-entropy.

Euclidean Loss For regression tasks, the Euclidean loss is often used. Defined as

$$E(\boldsymbol{x}, \boldsymbol{w}) = \sum \|g(\boldsymbol{x}, \boldsymbol{w}) - \boldsymbol{y}\|^2 \quad , \tag{3.114}$$

the Euclidean loss forms CNNs into a LS optimization. As discussed in Section 3.1.3, robust cost function can be used to reduce the influence of outliers.

[7] The hinge loss is defined $E(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i \neq c_g} \max\left(0, g_i(\boldsymbol{x}, \boldsymbol{w}) - g_{c_g}(\boldsymbol{x}, \boldsymbol{w}) + \Delta\right)$, with the ground-truth label c_g and margin Δ. The hinge loss accumulates penalty if the classification score for the correct label does not exceeds the Δ margin in comparison to any the wrong classes. In practice, it is sufficient to set $\Delta = 1$.

Regularization One important way to prevent networks from over-fitting is to add regularization for training. Two commonly used regularizations are the ℓ_1 norm and the ℓ_2 norm on parameters. Consider the data loss is given by $E_d(x, w)$, with ℓ_2 regularizer, the total loss takes the form

$$w^* = \arg\min_{w} E_d(x, w) + \eta E_r(w) = \arg\min_{w} E_d(x, w) + \eta w^T w \ , \tag{3.115}$$

where η is weight for balancing. The ℓ_2 regularizer is known as weight decay, which assumes the parameters have a zero-mean Gaussian prior. Note that without regularization, the original optimization is ill-posed. To see this, let w^* be the minimizer of $E_d(x, w)$, it is obvious the scaled version αw^* is also a minimizer. In this work, all the CNNs optimizations are regularized with ℓ_2.

3.3.2 State-of-the-art CNN Architectures

3.3.2.1 CNNs for Classification, Feature Extraction

LeNet proposed by **Lecun et al.** (1998) is widely recognized as the first CNN, which is developed to recognize handwritten digits. The LeNet architecture is very simple, which contains two convolutional layers, each followed by a pooling layer. Afterwards, two additional FC layers are used to output the classification of a single image. AlexNet developed by **Krizhevsky et al.** (2012) is the first deep CNN which made CNNs really popular, which was developed for ImageNet (**Russakovsky et al.** 2015) classification task and outperformed the existing algorithms by a significantly large margin. AlexNet is much deeper and larger than LeNet. It also presents a key difference to previous CNNs, instead of immediately followed by pooling layers, few convolutional layers were stacked together to introduce more nonlinearity. The next breakthrough came from the development of GoogleNet of **Szegedy et al.** (2015) and VGGNet by **Simonyan and Zisserman** (2015). The GoogleNet introduced the inception block, which significantly reduced the amount of parameters needed by AlexNet and gave better performance. The VGGNet has a consistent design, where all the convolutions have 3×3 spatial resolution. The original VGGNet contains over a hundred million parameters, where the first FC layer (fc6 in Table 3.1) contributes most of them. It is later found, most parameters in fc6 can be removed. With the CNNs get deeper, it was found that naively stacking convolutional layers degenerates rather than improve the performance of CNNs. One explanation is the arising difficulties in gradient back-propagation for very deep CNNs. To address this problem, **K. He et al.** (2016a) proposed the residual network (ResNet) which heavily uses identity mapping and batch normalization to help gradient back-propagation. The building block of ResNet is the residual unit, which describes the mapping from input to the output by $y^{(\ell+1)} = g^{(\ell)}\left(x^{(\ell)}, w^{(\ell)}\right) + x^{(\ell)}$. Followed by further development in **K. He et al.** (2016b), ResNet variants is successfully trained with 34-layer model up to 1000-layers model. Many ResNet based CNNs also produce the state-of-the-art performances for various benchmarks.

In this thesis, we developed algorithms based on VGG16 and ResNet101. The comparison of the network configuration as developed in the original works is presented in Table 3.1. For 1000-class image classification task, ResNet contains much more parameters, but yet significantly smaller amount of FLOPS and better accuracy than VGGNet. A detailed comparison of different CNNs and their accuracy on ImageNet is studied by **Canziani et al.** (2016).

3.3.2.2 CNNs for Dense Pixelwise Prediction

The aforementioned CNN architectures are developed for to produce a single output, *e.g.*, to classify one image as a whole. However, many applications require dense predictions, *e.g.*, image segmentation, flow estimation, depth prediction, generative modeling and etc. The regular CNNs typically downsample the input by a factor of 32 by pooling or large-stride convolutional layers. The downsampling is essential for CNNs to increase the receptive field, such that features can be extracted given the global context. One of the first CNNs to recover the full input resolution for dense prediction was proposed by **Long et al.** (2015), where transposed convolution was used for upsampling and up to $1/8$ input resolution was recovered. Later **Badrinarayanan et al.** (2017) and **Noh et al.** (2015) concurrently proposed the encoder-decoder CNNs. Based on VGGNet for feature extraction (referred as the encoder), the proposed network mirror the downsampling layers to perform upsampling. Additionally, the memorized unpooling is applied to reverse the pooling operations. With CNNs layers gets deeper and deeper as in ResNet, having a decoder as a mirror of encoder become less feasible. An alternative is to remove decoder by increasing the output resolution of CNNs to a reasonable level, such that the full resolution prediction can be achieved by regular image processing techniques. To this end, **L.-C. Chen et al.** (2018); **L.-C. Chen et al.** (2015) proposed the dilated VGGNet and dilated ResNet, which output prediction at $1/8$ of the input resolution using the dilated convolution. The output is then bilinear upsampled and refined by a fully-connected CRF (**Krähenbühl and Koltun** 2011). One limitation of the dilated CNNs is the significantly higher memory consumption to maintain the spatial resolution. Therefore, many works aim to design decoders based on regular CNNs (**Laina et al.** 2016; **G. Lin et al.** 2017; **Pohlen et al.** 2017).

One major focus of this thesis is the use of CNNs for semantic image segmentation. For this purpose, we have exploited the VGG16 based encoder-decoder architecture in Chapter 6 and Chapter 7, and further explore the ResNet101 based encoder-decoder architecture in Chapter 8.

3.3.3 Training

Deep Learning Framework There are many deep learning library alternatives. Some popular options include Caffe and Caffe2 (**Jia et al.** 2014), Torch (**Collobert et al.** 2002) and PyTorch (**Paszke et al.** 2017), Theano (**Bergstra et al.** 2010) and TensorFlow (**Abadi et al.** 2016). This thesis is developed upon Caffe and TensorFlow. Both libraries are written in C++ and CUDA. TensorFlow is a symbolic library, where CNNs are defined as computational

Table 3.1: The standard layer configuration of VGGNet by **Simonyan and Zisserman** (2015) and ResNet by **K. He et al.** (2016a) for 1000-class classification task, assuming the input resolution is 224×224. The notation $[a \times a, b] \times c$ reads as stacking c convolutional layers, each with b kernels of $a \times a$ spatial resolution.

layer name	output res.	VGG 16	VGG 19	layer name	output res.	ResNet 50	ResNet 101
conv1_x	224×224	$[3 \times 3, 64] \times 2$	$[3 \times 3, 64] \times 2$	conv1_x	112×112	$[7 \times 7, 64]$	$[7 \times 7, 64]$
maxpool1	112×112	$[2 \times 2]$, stride 2	$[3 \times 3, 64] \times 2$	maxpool1	56×56	$[2 \times 2]$, stride 2	$[2 \times 2]$, stride 2
conv2_x	112×112	$[3 \times 3, 64] \times 2$	$[3 \times 3, 64] \times 2$	conv2_x	56×56	$\begin{bmatrix} 1 \times 1, & 64 \\ 3 \times 3, & 64 \\ 1 \times 1, & 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, & 64 \\ 3 \times 3, & 64 \\ 1 \times 1, & 256 \end{bmatrix} \times 3$
maxpool2	56×56	$[2 \times 2]$, stride 2	$[2 \times 2]$, stride 2				
conv3_x	56×56	$[3 \times 3, 128] \times 3$	$[3 \times 3, 128] \times 4$	conv3_x	28×28	$\begin{bmatrix} 1 \times 1, & 128 \\ 3 \times 3, & 128 \\ 1 \times 1, & 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, & 128 \\ 3 \times 3, & 128 \\ 1 \times 1, & 512 \end{bmatrix} \times 4$
maxpool3	28×28	$[2 \times 2]$, stride 2	$[3 \times 3, 128] \times 2$				
conv4_x	28×28	$[3 \times 3, 256] \times 3$	$[3 \times 3, 256] \times 4$	conv4_x	14×14	$\begin{bmatrix} 1 \times 1, & 256 \\ 3 \times 3, & 256 \\ 1 \times 1, & 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, & 256 \\ 3 \times 3, & 256 \\ 1 \times 1, & 1024 \end{bmatrix} \times 23$
maxpool4	14×14	$[2 \times 2]$, stride 2	$[2 \times 2]$, stride 2				
conv5_x	14×14	$[3 \times 3, 512] \times 3$	$[3 \times 3, 512] \times 4$	conv5_x	7×7	$\begin{bmatrix} 1 \times 1, & 512 \\ 3 \times 3, & 512 \\ 1 \times 1, & 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, & 512 \\ 3 \times 3, & 512 \\ 1 \times 1, & 2048 \end{bmatrix} \times 3$
maxpool5	7×7	$[2 \times 2]$, stride 2	$[2 \times 2]$, stride 2				
fc_6	1×1	$[7 \times 7, 4096]$	$[7 \times 7, 4096]$	avgpool_2	1×1	$[7 \times 7]$, stride 1	$[7 \times 7]$, stride 1
fc_7	1×1	$[1 \times 1, 4096]$	$[1 \times 1, 4096]$				
output 1000-d fc, softmax							
Parameters		$13.8 \times 10e6$	$14.4 \times 10e6$			$25.6 \times 10e6$	$44.6 \times 10e6$
FLOPS		$15.3 \times 10e6$	$19.6 \times 10e9$			$3.8 \times 10e9$	$7.6 \times 10e9$

graphs. TensorFlow provides a good python interface and well supports distributed computation across multiple GPUs, whereas with Caffe, the python interface is very light-weighted but modification of the backend source code is easier.

Data Preprocessing: Augmentation, Normalization, Random Shuffling Providing limited training data, image augmentation attempts to prevent over-fitting by introducing noise into the training set with augmentation. Method exploited by this thesis includes resizing, cropping, flipping, gamma correction and contrast modification. At each iteration, for every individual image, these augmentation methods are performed with randomly chosen parameters given a predefined range.

Input normalization whitens the signals to be zero mean and unit variance, which consequentially improves the dynamics of optimization. One common normalization approach is to estimate the mean and variance for each input channel based on the entire training set and use these values at training and testing. When training with RGB or grayscale images, an alternative is to apply the transform $(2x/255.0 - 1.0)$ for each pixel in each channel. The later is used in this thesis.

Data shuffling is another important tip for network training, which is especially important for stochastic gradient descent and training with small datasets. Shuffling prevents CNNs tuning towards the correlations from observing the same mini batches with the same order repeatedly. In our training, the order of images are shuffled after every epoch.

Weight Initialization versus Transfer Learning Training CNNs is a process of solving highly non-convex optimizations with a huge number of parameters, where weight initialization is very important. Consider the input data are properly normalized to be zero-mean, it is reasonable to assume weights will be zero-centered Gaussian distribution. Based on this assumption, the naive initialization for convolutional kernel is to sample from Gaussian distributions $w \sim \mathcal{N}(0, 1)$. The problem with such initialization is that the gradient of layers with different amount of neurons will be different. To address this problem, **Glorot and Bengio** (2010) proposed to scale the zero-mean unit Gaussian by $2/(n_i + n_o)$, where n_i, n_o are the number of input and output neurons of the convolutional layer. Later, **K. He et al.** (2015) proposed the proper scale should be $2.0/\sqrt{n_\ell}$, where n_ℓ is the total number of weights contained in a convolutional kernel.

The above described weights initialization techniques refer to training CNNs from scratch. In practice, training from scratch is very demanding, which requires a lot data, hardware support and careful parameter tuning. Therefore, it is more common and beneficial to initialize weights from pre-trained models and perform transfer learning, also refer as fine-tuning. When the data distribution between original dataset used to obtain the pre-trained model and the current dataset are similar, transfer learning is more likely to be beneficial. In our work, we always fine-tuned from CNN models pre-trained on ImageNet.

PART II

Selected Publications

Equations are just the boring part of mathematics. I attempt to see things in terms of geometry.

– Stephen Hawking

CHAPTER 4

Planar Simplification and Texturing of Dense Point Cloud Maps

ABSTRACT Dense RGB-D based SLAM techniques and high-fidelity LIDAR scanners are examples from an abundant set of systems capable of providing multi-million point datasets. These large datasets quickly become difficult to process and work with due to the sheer volume of data, which typically contains significant redundant information, such as the representation of planar surfaces with hundreds of thousands of points. In order to exploit the richness of information provided by dense methods in real-time robotics, techniques are required to reduce the inherent redundancy of the data. In this paper we present a method for efficient triangulation and texturing of planar surfaces in large point clouds. Experimental results show that our algorithm removes more than 90% of the input planar points, leading to a triangulation with only 10% of the original amount of triangles per planar segment, improving upon an existing planar simplification algorithm. Despite the large reduction in vertex count, the principal geometric features of each segment are well preserved. In addition to this, our texture generation algorithm preserves all color information contained within planar segments, resulting in a visually appealing and geometrically accurate simplified representation.

4.1 Introduction

The generation of 3D models of real-world environments is of high interest to many application fields including professional civil engineering, environment-based game design,

© 2013 IEEE. Reprinted, with permission, from Lingni Ma, Thomas Whelan, Egor Bondarev, Peter H. N. de With and John McDonald, *Planar Simplification and Texturing of Dense Point Cloud Maps*, in proc. European Conference on Mobile Robots (ECMR), Sept 2013. Revised layout, minor adaptations and extended visualization for Figure 4.1.

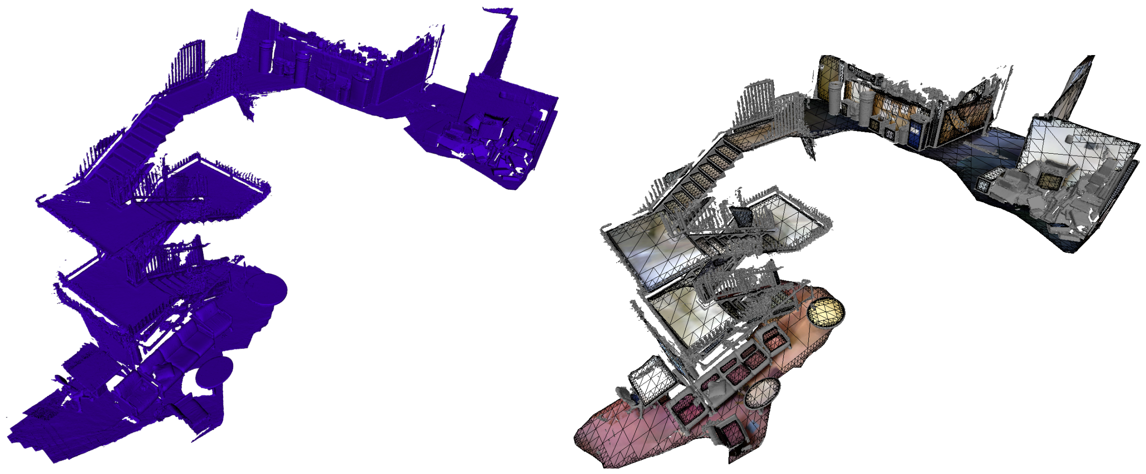


Figure 4.1: Scene triangulation showing a simplified mesh (right) for planar segments with non-planar features highlighted, as opposed to the dense reconstruction by traditional method (left).

3D printing and robotics. Industrial Light Detection And Ranging (LIDAR) platforms and extended scale RGB-D mapping systems can output dense high-quality point clouds, spanning large areas that contain millions of points (**Henry et al.** 2012; **Whelan et al.** 2012). Key issues with such large-scale multi-million point datasets include difficulties in processing the data within reasonable time and a high memory requirement. In addition to this, some features of real-world maps, such as walls and floors, end up being over-represented by thousands of points when they could be more efficiently and intelligently represented with geometric primitives. In particular the use of geometric primitives to represent a large 3D map to localize against is emerging as a feasible means of robot localisation (**Fallon et al.** 2012). In this paper, we examine the problem of planar surface simplification in large-scale point clouds with a focus on quality and computational efficiency.

4.2 Related Work

In the literature triangular meshing of 3D point clouds is a well-studied problem with many existing solutions. One class of triangulation algorithms computes a mathematical model prior to triangulation to ensure a smooth mesh while being robust to noise (**Jalba and Roerdink** 2009; **Kazhdan et al.** 2006). This type of algorithm assumes surfaces are continuous without holes, which is usually not the case in open scene scans or maps acquired with typical robotic sensors. Another class of algorithms connects points directly, mostly being optimized for high-quality point clouds with low noise and uniform density. While these algorithms retain fine details in objects (**Bernardini et al.** 1999; **Scheidegger et al.** 2005), they are again less applicable to noisy datasets captured with an RGB-D or LIDAR sensor, where occlusions create large discontinuities.

With real-world environment triangulation in mind, the Greedy Projection Triangulation

(GPT) algorithm has been developed (**Gopi and Krishnan** 2002; **Marton et al.** 2009). The algorithm creates triangles in an incremental mesh-growing approach, yielding fast and accurate triangulations. However, the GPT algorithm keeps all available points to preserve geometry, which is not always necessary for point clouds containing surfaces that are easily approximated by geometric primitives. To solve this problem a hybrid triangulation method was developed by **Ma et al.** (2013c), where point clouds are segmented into planar and non-planar regions for separate triangulation. The QuadTree-Based (QTB) algorithm was developed to decimate planar segments prior to triangulation. The QTB algorithm significantly reduces the amount of redundant points, although a number of limitations degrade its performance. For example, the algorithm does not guarantee that final planar points will lie inside the original planar region, which can lead to noticeable shape distortion. The algorithm also produces duplicate vertices, overlapping triangles and artificial holes along the boundary.

To summarize, existing triangulation algorithms perform poorly in removing redundancy in dense point clouds, or are not suited to the kind of data typically acquired with common robotic sensors. In this paper we address these problems with two main contributions based on the work of **Ma et al.** (2013c). Firstly we present an accurate and robust algorithm for planar segment decimation and triangulation. In comparison to the existing QTB algorithm, our algorithm guarantees geometrical accuracy during simplification with fewer triangles, without duplicate points, artificial holes or overlapping faces. Secondly we present a method to automatically generate textures for the simplified planar mesh based on dense colored vertices. Our experimental results show that the presented solutions are efficient in processing large datasets, through the use of a multi-threaded parallel architecture.

4.3 System Overview

4.3.1 Building Blocks

Our system architecture is shown in Figure 4.2. It takes a point cloud as input and generates a triangular mesh as output. If the input is a colored point cloud, the output can also be a textured 3D model. The processing pipeline consists of three main blocks.

Plane Detection segments the input into planar and non-planar regions to enable separate triangulation and parallel processing. This design is especially beneficial for real-world environments, where multiple independent planar surfaces occur frequently. In our system we apply a local curvature-based region growing algorithm for plane segmentation by **Ma et al.** (2013c).

Non-Planar Segment Triangulation generates a triangular mesh for non-planar segments using the GPT algorithm by **Marton et al.** (2009). Given a colored point cloud, we preserve the color information for each vertex in the output mesh. Dense triangular meshes with colored vertices can be rendered (with Phong interpolation) to appear similar to textured

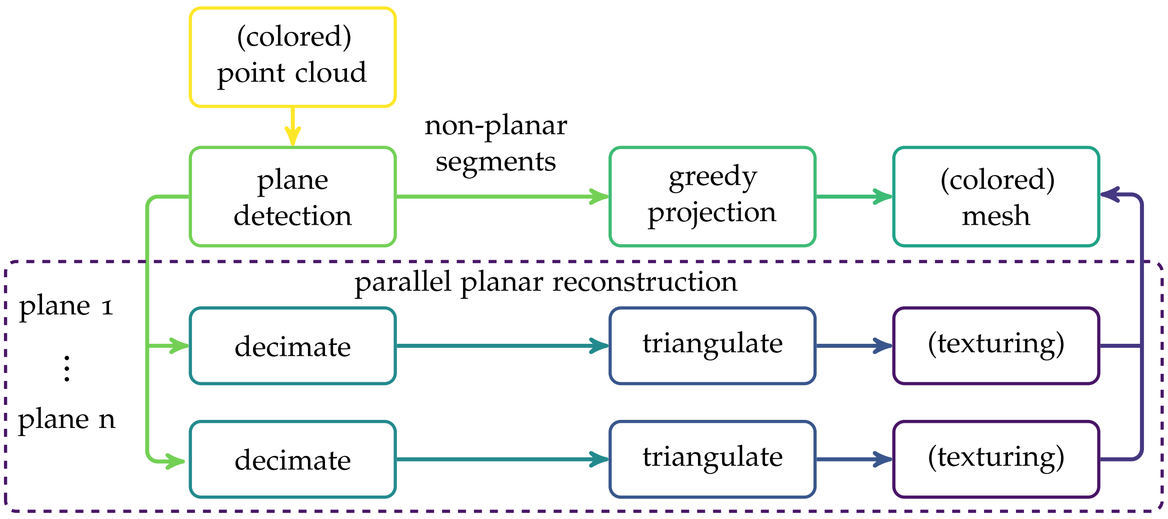


Figure 4.2: Parallel system architecture to process point clouds of large-scale open scene scans or maps.

models. Additionally, as opposed to using textures, maintaining color in vertices of non-planar segments provides easier access to appearance information for point cloud based object recognition systems.

Planar Segment Triangulation triangulates planar segments and textures the mesh afterwards, if given a colored point cloud. In our system we improve the decimation algorithm by **Ma et al.** (2013c) and further develop a more accurate and robust solution for triangulation. A detailed description of our algorithm is provided in Section 4.4. Our method for planar segment texture generation is described in Section 4.5.

4.3.2 Computationally Efficient Architecture

To improve computational performance, a multi-threaded architecture is adopted, exploiting the common availability of multi-core CPUs in modern hardware. We apply a coarse-grained parallelization strategy, following the Single Program Multiple Data (SPMD) model (**Darema et al.** 1988). Parallel triangulation of planar segments is easily accomplished by dividing the set of segments into subsets that are distributed across a pool of threads. For maximum throughput of the entire pipeline, segmentation and triangulation overlap in execution. With an n-core CPU, a single thread is used for segmentation and the remaining $n-1$ threads are used for triangulation, each with a queue of planar segments to be processed. Upon segmentation of a new planar region, the segmentation thread checks all triangulation threads and assigns the latest segment to the thread with the lowest number of points to be processed. This strategy ensures even task distribution among all threads. When plane segmentation is finished, the segmentation thread begins the non-planar trian-

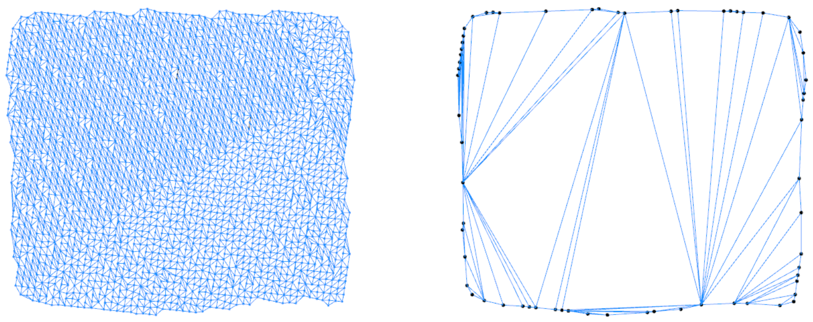


Figure 4.3: Undesirable planar triangulation: the left GPT mesh over-represents the shape while the right boundary-based Delaunay triangulation produces unnatural skinny triangles.

gulation in parallel to the other triangulation threads.

4.4 Triangulation of Planar Segments

In this section, our algorithm for planar segment decimation and triangulation is described. A simplified mesh of a planar segment is generated by removing redundant points that fall within the boundary of the segment. In the following text the input planar segment is denoted as $\mathcal{P}$, made up of points $\mathbf{p} \in \mathbb{R}^3$. With colored point clouds, each point $\mathbf{p}$ also contains (R, G, B) color components.

4.4.1 QuadTree-Based Decimation

Planar segments have a simple shape which can be well described by points on the boundary of segment. Interior points only add redundancy to the surface representation and complicate the triangulation results. Figure 4.3 shows such an example, where the planar segment is over-represented with thousands of triangles generated with the GPT algorithm using all planar points. However, a naïve solution to remove all interior points and triangulate only with boundary points normally leads to skinny triangles, again shown in Figure 4.3. With these observations in mind, the quadtree proves to be a useful structure to decimate the interior points of a segment while preserving all boundary points for shape recovery (**Ma et al.** 2013c).

4.4.1.1 Preprocessing

To prepare a planar segment for decimation it is first denoised and aligned to the x-y axes. We employ Principal Component Analysis (PCA) over the planar segment to compute a least-squares plane fit as well as an affine transformation T for x-y axes alignment. The aligned planar segment is denoted as $\mathcal{P}_t$. Afterwards, the boundary points of $\mathcal{P}_t$ are extracted as an α-shape (**Duckham et al.** 2008; **Pateiro-López and Rodríguez-Casal** 2010).

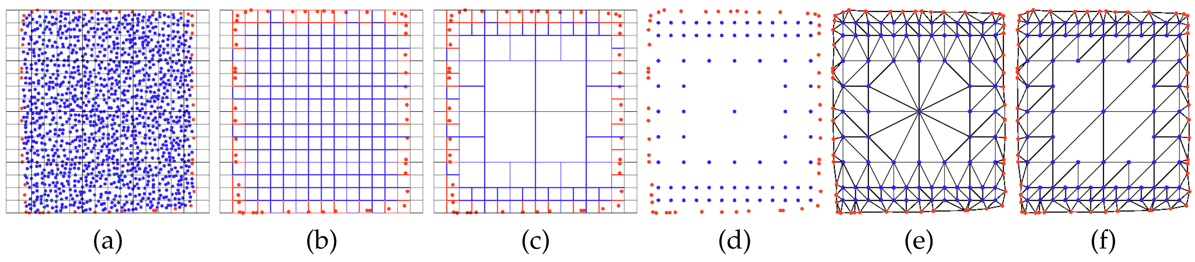


Figure 4.4: Planar decimation and triangulation (boundary and interior points are red and blue, respectively). The sub-figures: (a) initialize by subdividing the quadtree bounding box; (b) classify nodes into interior (blue), boundary (red) and exterior (black); (c) merge interior nodes; (d) generate vertices; (e) point-based triangulation; (f) polygon-based triangulation.

We denote the boundary as the concave hull $\mathcal{H}$ of the planar segment, which is an ordered list of vertices describing a polygon for which $\forall \mathbf{p} \in \mathcal{P}_t$ and $\mathbf{p} \notin \mathcal{H}$, $\mathbf{p}$ is inside the polygon.

4.4.1.2 Decimation

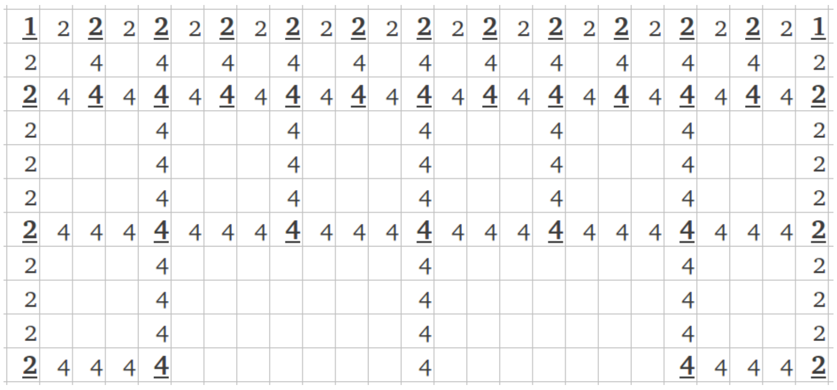
Planar segment point decimation consists of four steps as shown in Figure 4.4. Firstly, a quadtree is constructed by subdividing the bounding box of $\mathcal{P}_t$ into a uniform grid of small cells. Typically the 2D bounding box is non-square, in which case the smallest side is extended to equalize width and height. The resulting bounding box $\mathbf{b}$ is composed of a minimum point $\mathbf{b}_{min}$ and a maximum point $\mathbf{b}_{max}$, with a dimension $\mathbf{s} = \mathbf{b}_{max} - \mathbf{b}_{min}$. Secondly, the quadtree nodes are classified as either interior, boundary or exterior. An *interior* node is fully contained within the polygon $\mathcal{H}$, while an *exterior* node is fully outside. All others are *boundary* nodes, which intersect $\mathcal{H}$. Thirdly, the interior nodes of the quadtree are merged to create nodes of variable size, typically largest around the center and becoming increasingly fine-grained when approaching the boundary. When a parent node contains only interior children, the four children nodes are merged into one. The merged node is then also classified as interior, allowing further recursive merging with its siblings. Finally, the corner points of the remaining interior nodes are extracted as the new internal vertices $\mathcal{I}$ of $\mathcal{P}_t$, while all boundary points $\mathcal{H}$ are preserved.

4.4.2 Triangulation

We provide two methods for triangulation of a simplified planar segment: 1) a low-complexity Point-Based Triangulation and 2) an alternative Polygon-Based Triangulation. Both methods make use of the Constrained Delaunay Triangulation (CDT) by **Domiter and Zalik** (2008).

4.4.2.1 Point-Based Triangulation

The point-based approach is a low-complexity triangulation method, where CDT is directly applied to the decimated segment. The ordered boundary vertices $\mathcal{H}$ serve as constraining



1	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**1**
2		4		4		4		4		4		4		4		4		4		4		4		2
2	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**2**
2				4				4				4				4				4				2
2				4				4				4				4				4				2
2				4				4				4				4				4				2
2	4	4	4	**4**	4	4	4	**4**	4	4	4	**4**	4	4	4	**4**	4	4	4	**4**	4	4	4	**2**
2				4								4								4				2
2				4								4								4				2
2				4								4								4				2
2	4	4	4	**4**								4								**4**	4	4	4	**2**

Figure 4.5: Degree grid of the upper planar segment in Figure 4.4 (0-valued cells hidden). The underlined bold values are the degrees of the inner vertices $\mathcal{I}$.

edges and the inner vertices $\mathcal{I}$ are used as input points. An example output is shown in Figure 4.4. Point-based triangulation has all of the advantages of Delaunay triangulation but does produce more triangles than the polygon-based approach described next.

4.4.2.2 Polygon-Based Triangulation

The regular grid pattern of the inner vertices $\mathcal{I}$ immediately lends itself to a simple triangulation strategy, where two right-angled triangles are created over each interior node of the merged quadtree. To complete the triangulation, the space between the interior right-angled triangles and the boundary points $\mathcal{H}$ is triangulated using CDT. Two sets of constraining edges are input to the CDT, one being $\mathcal{H}$ and the other being a rectilinear isothetic polygon that bounds interior triangles. This two-step triangulation is similar to the QTB algorithm by **Ma et al.** (2013c). However, a major difference lies in how the boundary points are connected. With our CDT-based approach, we avoid overlapping triangles and artificial holes that would normally be produced by the QTB algorithm.

Efficient computation of the polygon which exactly bounds the interior vertices $\mathcal{I}$ is non-trivial, since the interior nodes provide only sparse spatial information for geometric operations. We invoke a solution that maps the interior vertices onto a binary image, where the bounding polygon can be easily extracted using a greedy nearest-neighbour tracing algorithm normally used in image processing (**Marquegnies** 2011).

The binary image is represented by an $n \times n$ array, where $n = 2^{d+1} + 1$ and d is the quadtree depth. This provides a 2D grid large enough to represent empty space between the two vertices of any edge. To project a vertex $\mathbf{v} \in \mathcal{I}$ onto the array, a mapping function $f : \mathbb{R}^3 \to \mathbb{N}^2$ is defined by

$$f(\mathbf{v}) = \frac{n(\mathbf{v} - \mathbf{b}_{min})}{\mathbf{s}}, \tag{4.1}$$

where $\mathbf{b}$ is the bounding box and $\mathbf{s}$ is its dimension. The division is performed on an element-by-element basis. Given that $\mathcal{I}$ is aligned to the x-y axes, function f effectively maps

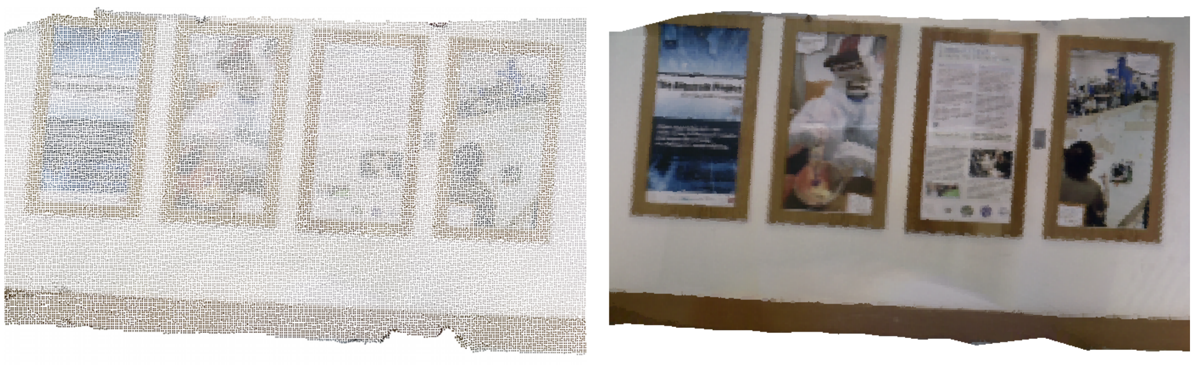


Figure 4.6: Texture generation: (left) plane segment from a colored point cloud; (right) generated texture.

from $\mathbb{R}^2$ to $\mathbb{N}^2$. We associate two elements with each array cell: a reference to the mapped vertex (effectively implementing f^{-1}) and a degree value to quantify vertex connectivity. Initially, the degree is zero for all cells. During the triangulation of $\mathcal{I}$, the degree grid is populated. When a vertex is extracted from the merged quadtree, the reference of the corresponding cell is updated and its degree is increased by 1. This policy alone cannot fully recover the degree of a given vertex, since only the two ends of an edge are obtained from quadtree vertices. To overcome this problem, all cells between the two ends of an edge also have their degree increased by 2. Figure 4.5 shows a part of the degree grid of the planar segment in Figure 4.4. If we consider the interior triangulation to be a graph, the 2D degree grid resolves the degree of each vertex. All non-zero cells are treated as "1-valued" foreground pixels and the rest as "0-valued" background pixels in the binary image representation.

4.5 Texture Generation

In this section we present our texture generation algorithm for planar segments using dense colored point clouds. Due to the significant loss of colored vertices during decimation, the appearance of a simplified planar segment is greatly diminished. We therefore generate textures prior to decimation for the purpose of texture mapping the simplified planar mesh.

We generate textures by projecting the vertex colors of the dense planar segment onto a 2D RGB texture $\mathcal{E}(x, y) \in \mathbb{N}^3$. We define a texture resolution $\mathbf{d}$ as some resolution factor r times $\mathbf{s}$, where $\mathbf{s}$ assumes the dimension of the bounding box $\mathbf{b}$. In our experiments a value of $r = 100$ provides good-quality textures. The resolution factor can also be automatically computed based on point cloud density. Each pixel $\mathbf{a} \in \mathcal{E}$ is first mapped to a 3D point $\mathbf{v}$ by a mapping function $g : \mathbb{N}^2 \to \mathbb{R}^3$, defined as

$$g(\mathbf{a}) = \frac{\mathbf{a}\mathbf{s}}{\mathbf{d}} + \mathbf{b}_{min}, \tag{4.2}$$

ALGORITHM 4.1: Vertex color to texture.

```
Input: $\mathcal{P}_t$ set of transformed input vertices
Input: $\mathcal{H}$ concave hull of $\mathcal{P}_t$
Output: $\mathcal{E}$ 2D RGB texture
1 foreach pixel $\mathbf{p}$ in $\mathcal{E}$ do
2     $\mathbf{v} \leftarrow g(\mathbf{p})$;
3     if $\mathbf{v}$ is inside $\mathcal{H}$ then
4         $\mathbf{n} \leftarrow$ nearest-neighbour of $\mathbf{v}$ in $\mathcal{P}_t$;
5         $\mathbf{p} \leftarrow (\mathbf{n}_R, \mathbf{n}_G, \mathbf{n}_B)$;
6     else
7         $\mathbf{p} \leftarrow (0, 0, 0)$;
```

ALGORITHM 4.2: uv texture coordinate calculation.

```
Input: $\mathcal{O}$ set of final face vertices
Output: $\mathcal{U}$ $uv$ texture coordinates for $\mathcal{O}$
1 foreach vertex $\mathbf{v}$ in $\mathcal{O}$ do
2     $\mathbf{a} \leftarrow g^{-1}(\mathbf{v})$;
3     $u \leftarrow \frac{\mathbf{a}_x}{\mathbf{d}_x}$;
4     $v \leftarrow 1.0 - \frac{\mathbf{a}_y}{\mathbf{d}_y}$;
5     Add $(u, v)$ to $\mathcal{U}$;
```

with an element-by-element calculation. Since $\mathcal{P}_t$ is aligned to the x-y axes, the function g effectively maps to $\mathbb{R}^2$. A corresponding colored point to $\mathbf{v}$ in $\mathcal{P}_t$ is found by a nearest neighbor search using a kd-tree. We have chosen this approach as it produces good-quality textures while being computationally inexpensive. However, it can be easily extended to produce even higher-quality textures by averaging a number of k-nearest neighbours. Algorithm 4.1 describes the texture generation process. Figure 4.6 shows an input planar segment and the output texture.

When texture mapping the final planar mesh, the uv texture coordinates $\mathcal{U}$ for the vertices $\mathcal{O}$ of each face are computed with the inverse function $g^{-1} : \mathbb{R}^3 \to \mathbb{N}^2$, derived from Equation (4.2) as

$$g^{-1}(\mathbf{v}) = \frac{\mathbf{d}(\mathbf{v} - \mathbf{b}_{min})}{\mathbf{s}}. \tag{4.3}$$

With x-y axes aligned points, g^{-1} is actually mapping from $\mathbb{R}^2$. Algorithm 4.2 describes the uv-coordinates computation. The list $\mathcal{U}$ guarantees a 1-to-1 mapping to the set $\mathcal{O}$.

Any objects lying on a planar segment are completely excluded from the texture and not projected onto the plane. In fact, the generated texture implicitly provides the Voronoi diagram of the face of the object lying on any plane, which in turn provides position and orientation information of any object lying on a segmented plane, as shown in Figure 4.8.

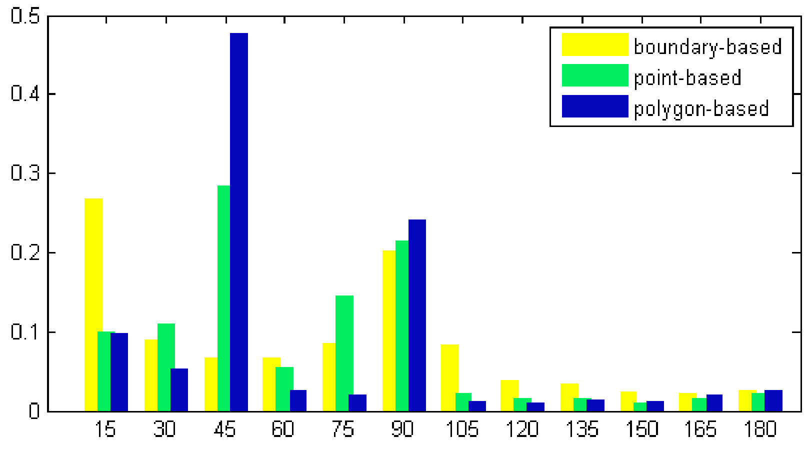


Figure 4.7: Triangulation quality measured with the angle distribution of planar meshes.

Table 4.1: Planar point reduction with our decimation algorithm in comparison to the QTB algorithm.

	data 1	data 2	data 3	data 4
Total points	890,207	1,094,910	2,799,744	5,641,599
Planar points	540,230	708,653	1,303,012	2,430,743
QTB decimation	105,663	303,348	189,772	457,678
Our decimation	47,457	84,711	43,257	76,624

4.6 Evaluation and Results

In this section we evaluate our work with a series of experiments. We ran our C++ implementation on Ubuntu Linux 12.04 with an Intel Core i7-3930K CPU at 3.20 GHz with 16 GB of RAM. Four colored point clouds of real-world environments were used in the experiments, as shown in Figure 4.10 (a). These datasets encompass a wide variation in the number of points, planar segments and their geometry. All four datasets have been acquired with an implementation of the Kintinuous dense RGB-D mapping system by **Whelan et al. (2013)**.

4.6.1 Triangulation Performance

To assess the triangulation performance, qualitative and quantitative evaluations are presented. A comparison of the triangulation algorithms is shown in Figure 4.10. Additionally we present map fly-throughs in a video submission available at `http://www.youtube.com/watch?v=dn6ccmFXSzQ` . It can be seen that both algorithms produce a highly simplified triangulation, while preserving the principal geometry of the segment.

Further assessment of mesh quality is done by measuring the angle distribution across meshes. A naïve simplified planar mesh is set as a baseline, which applies Delaunay tri-

Table 4.2: Planar mesh simplification with our triangulation algorithms measured with triangle counts, in comparison to GPT and the QTB algorithm.

	data 1	data 2	data 3	data 4
GPT	1,020,241	1,350,004	2,293,599	4,665,067
QTB	90,087	288,833	182,648	433,953
Point-based	85,653	161,270	79,988	143,396
Polygon-based	76,367	130,642	66,413	118,275

angulation to only the boundary points of a planar segment. The normalized distribution is shown in Figure 4.7, collected from the 400 planar segments of the four datasets. It can be seen that approximately 80% of the triangles from the polygon-based triangulation are isosceles right-angle triangles, resulting from the quadtree-based triangulation. With point-based triangulation, the angles spread over 30°-90°, whereas the naïve boundary-based triangulation shows an even more random distribution. Defining a skinny triangle as one with a minimum angle $<15°$, the percentages of skinny triangles with boundary-based, point-based and polygon-based triangulation are 28%, 10% and 10%, respectively.

The effectiveness of planar segment decimation is also evaluated. Table 4.1 shows the point count for planar point decimation. Approximately 90% of the redundant points are removed with our algorithm, which is 15% more than the QTB algorithm, despite the fact that both algorithms are based on a quadtree. Part of this reduction gain comes from our triangulation methods, which add no extra points once decimation is done, unlike the QTB algorithm. In Table 4.2, the mesh simplification statistics with triangle counts are also given. We take the triangle count of GPT for non-decimated planar segments as the baseline. In accordance with the point count reduction, both of our algorithms require no more than 10% of the amount of triangles of a non-decimated triangulation, and both perform better than the QTB algorithm.

4.6.2 Texture Generation Performance

In Figure 4.10, generated textures are shown. The output textures incorporate almost all visual information contained in the original dense point cloud, enabling a photo-realistic and aesthetically-pleasing textured 3D model.

4.6.3 Computational Performance

Lastly, we evaluate the computational efficiency of our algorithms and the parallel system for large-scale data processing. The baseline for comparison is standard serial processing with the GPT and QTB algorithms. Table 4.3 shows the execution times. The point-based and polygon-based triangulations are approximately of the same speed, both 2 to 3 times faster than the GPT and QTB algorithms. The results also show that the texture generation algorithm is fast in execution, processing multi-million point datasets in less than 2 seconds.

Table 4.3: Efficiency of triangulation and the parallel architecture, measured in seconds. The 1:x ratio denotes 1 segmentation thread with x triangulation threads.

	data 1	data 2	data 3	data 4
Number of planar segments	101	116	66	117
Serial GPT	18.6	24.3	44.2	91.1
Serial QTB	16.7	18.7	38.3	73.1
Serial point-based	6.9	9.8	17.7	40.2
Serial polygon-based	6.9	9.5	17.8	40.0
Serial polygon-based (texture)	8.3	10.0	20.3	41.4
1:1 Polygon-based	6.4	8.1	15.1	33.8
1:1 Polygon-based (texture)	7.6	8.5	17.4	35.2
1:3 Polygon-based	3.6	4.2	8.3	19.2
1:3 Polygon-based (texture)	4.4	4.1	9.2	19.6
1:5 Polygon-based	3.7	3.5	7.9	16.1
1:5 Polygon-based (texture)	4.7	3.5	8.7	16.2

Examining the bottom half of Table 4.3, it is clear that the parallel system architecture has a profound effect on the overall performance. The execution time decreases with an increasing number of triangulation threads. An effect of diminishing returns becomes apparent as the number of triangulation threads increases, due to the overhead associated with the parallel implementation. However, as the per-thread workload increases, such as inclusion of texture generation, the overhead of parallelization becomes overshadowed.

4.6.4 Discussion

Both point-based and polygon-based triangulation yield accurate and computationally efficient planar segment triangulations with significant point and triangle count reductions, both exceeding the performance of the QTB algorithm. The point-based approach is of low complexity and maintains good triangular mesh properties that are desirable for lighting and computer graphics operations. The polygon-based approach yields higher point and triangle count reductions with a more regularized mesh pattern, capturing information about the scene in the form of principal geometric features, such as the principal orientation of a planar segment. While the polygon-based method produces less triangles, it does generate T-joints in the mesh. Such feature is detrimental when employing Gouraud shading and other lighting techniques to render a mesh with colored vertices. The polygon-based and point-based methods offer a trade-off depending on the desired number of triangles or the intended use of the final triangulation. With robot navigation in mind, the low polygon-count models achieved with our system are suitable for use in a primitives-based localization system, such as the KMCL system of **Fallon et al.** (2012).

The gaps between planar and non-planar triangulations are apparent. The gap can also

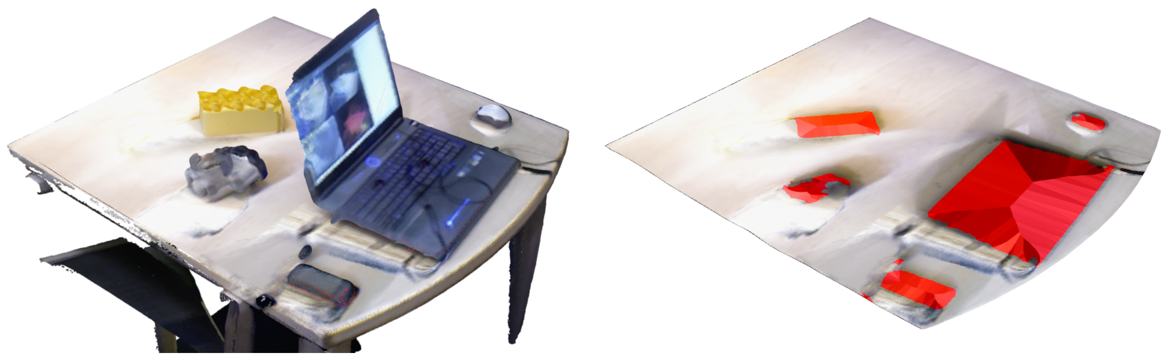


Figure 4.8: Implicit object information from texture generation, from left to right: (a) input colored point cloud; (b) generated texture with implicit Voronoi diagrams and locations of objects resting on the plane highlighted.

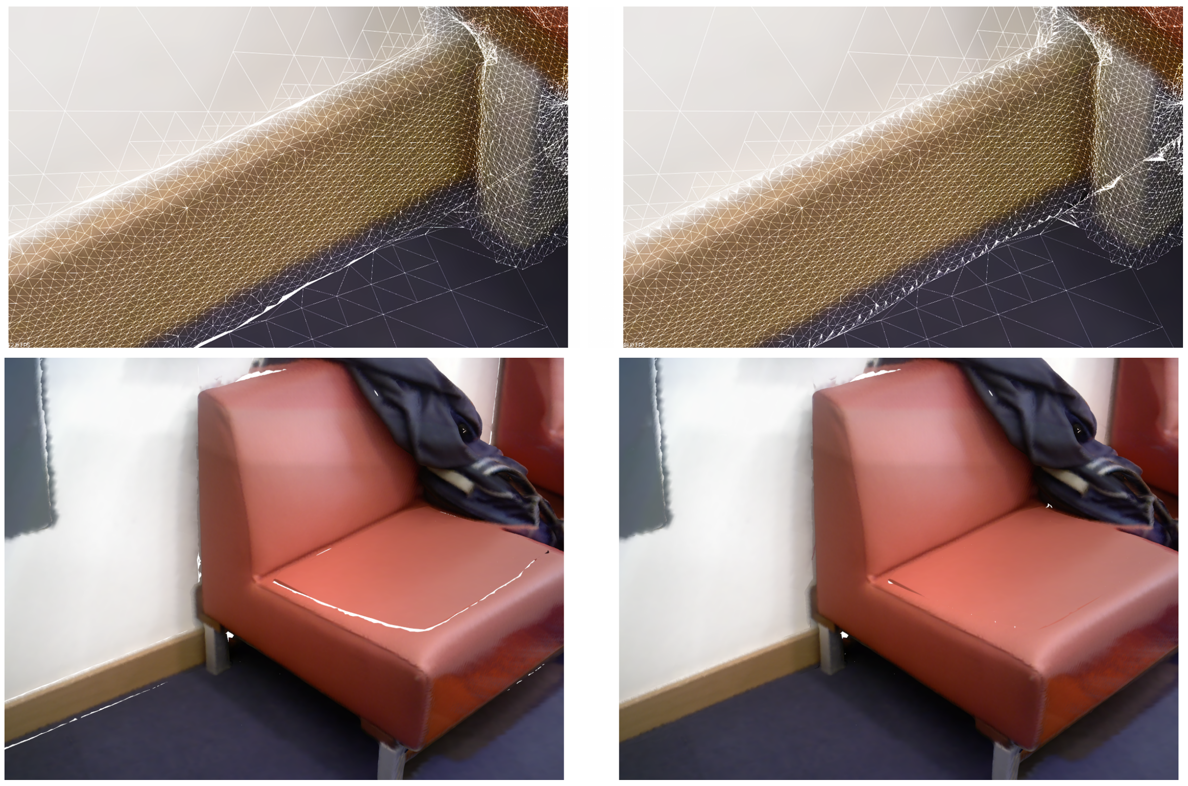
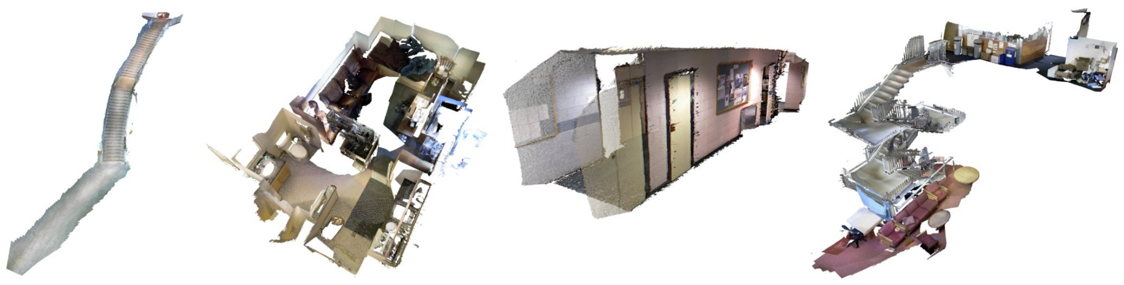


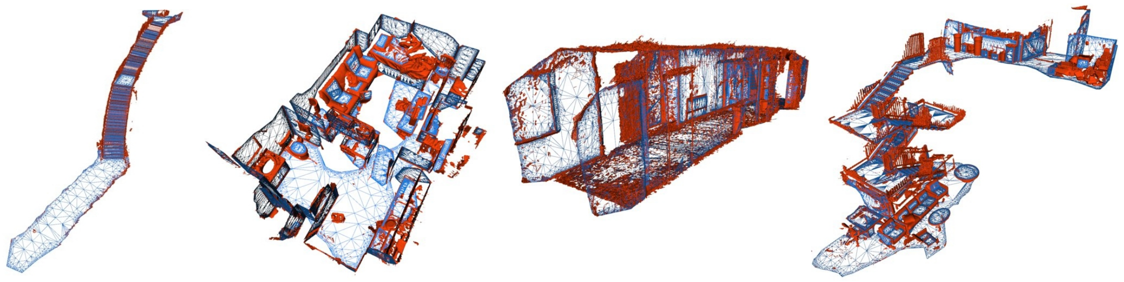
Figure 4.9: Joining of GPT mesh with planar segment triangulations. Left shows unjoined segments and right shows segments joined with interpolated boundary vertices.

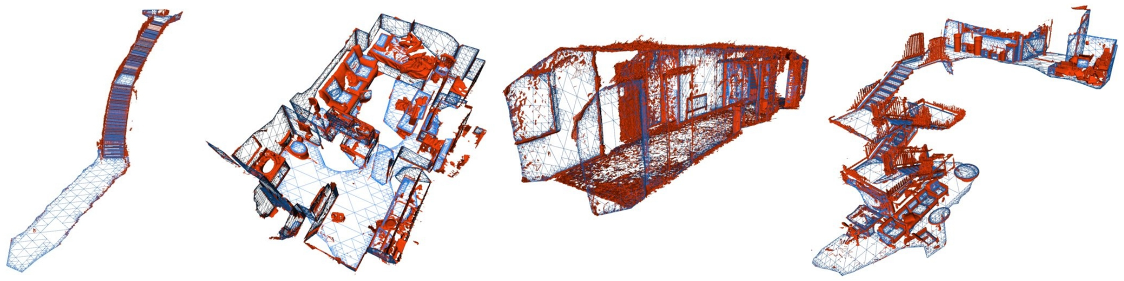
be closed by including the boundary vertices of the segmented planes into the non-planar segment GPT triangulation, as shown in Figure 4.9. The number of boundary vertices can be increased with a smaller alpha value when computing the concave hull of each segment or by linearly interpolating between boundary vertices. Extra vertices can also be extracted from the vertex degree grid used in polygon-based triangulation. In our system we chose to leave these gaps open, as this separation gives an easier visual understanding of any map, implicitly providing a separation between structural features (like walls, table tops) and "object" features, useful in automatic scene understanding, manipulation and surface classification.

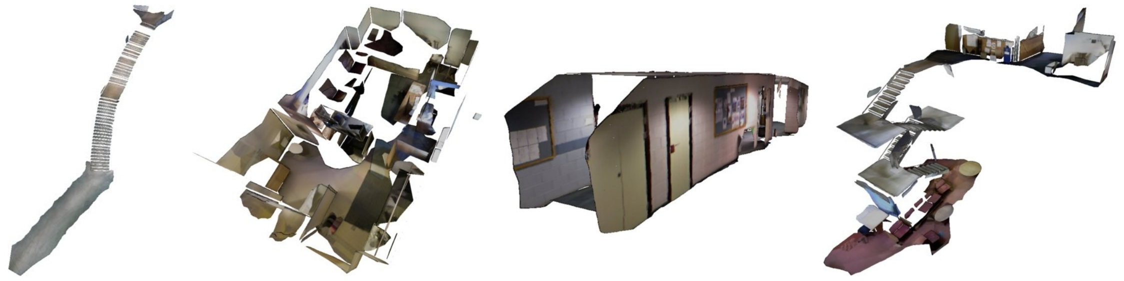
4.7 Conclusions

In this paper we have studied the problem of triangulation of planar segments from dense point clouds with a focus on quality and efficiency. Three significant contributions are made. Firstly, we have made a strong improvement on planar segment reconstruction. Both of the presented point-based and polygon-based triangulation methods produce a more accurate, simpler and robust planar triangulation than the existing QTB algorithm. With these two algorithms approximately 90% of input planar points are removed, and the planar segments are triangulated with no more than 10% of the amount of triangles required without decimation. Secondly, we have developed a computationally inexpensive algorithm to automatically generate high-quality textures for planar segments based on colored point clouds. Last, our parallel system multi-threaded architecture enhances the efficiency in processing large-scale datasets. The results show that our system provides a computationally manageable map representation for real-world environment maps and also generates a visually appealing textured model in a format useful for real-time robotic systems.


(a) Four dense colored point clouds used for evaluation, numbered 1 to 4 from left to right.


(b) Point-based triangulation: planar and non-planar meshes shown in blue and red.


(c) Polygon-based triangulation: planar and non-planar meshes shown in blue and red.


(d) Textured simplified planar segments from each dataset.

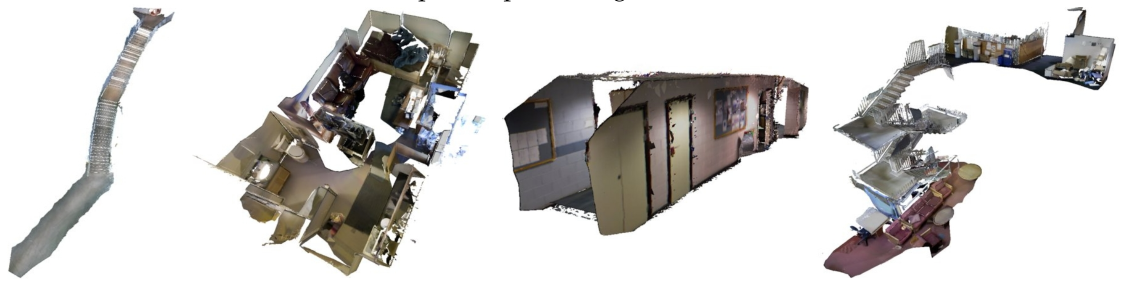

(e) Complete 3D model with our proposed system.

Figure 4.10: Four evaluated datasets (numbered 1 to 4 from left to right) with various triangulation results and texturing results.

CHAPTER 5

CPA-SLAM: Consistent Plane-Model Alignment for Direct RGB-D SLAM

ABSTRACT Planes are predominant features of man-made environments which have been exploited in many mapping approaches. In this paper, we propose a real-time capable RGB-D SLAM system that consistently integrates frame-to-keyframe and frame-to-plane alignment. Our method models the environment with a global plane model and – besides direct image alignment – it uses the planes for tracking and global graph optimization. This way, our method makes use of the dense image information available in keyframes for accurate short-term tracking. At the same time it uses a global model to reduce drift. Both components are integrated consistently in an expectation-maximization framework. In experiments, we demonstrate the benefits our approach and its state-of-the-art accuracy on challenging benchmarks.

5.1 Introduction

Man-made environments are composed of many objects with simple geometric properties that can be used for reference in visual simultaneous localization and mapping (SLAM) systems. In this process, these objects represent higher-level semantic information in the map, which could be used, *e.g.,* for robots to reason about the world, to communicate information, or to display information to users, for instance, using virtual reality devices. The object states can be concurrently estimated with the motion of the camera. Integrated consistently, camera motion and object state estimation can benefit from each other.

© 2016 IEEE. Reprinted, with permission, from Lingni Ma, Christian Kerl, Jörg Stückler and Daniel Cremers, *CPA-SLAM: Consistent Plane-model Alignment for Direct RGB-D SLAM*, in proc. of IEEE International Conference on Robotics and Automation (ICRA), May 2016. Revised layout, minor adaptations and extended visualization for Figure 5.1, Figure 5.6.

Figure 5.1: Demo of the CPA SLAM algorithm. First row: plane segmentation and association of all keyframes (minimal overlapping required), which contains false detections, *e.g.,* objects on the table. Second row: the average soft labeling of plane from tracking with an EM framework. The probability of plane drops from yellow to blue. Third Row: the final mapping and the plane-keyframe constraints for graph optimization. Last row: colored model and the camera trajectories before (red) and after (blue) optimization.

In this paper, we propose a novel formulation for including planes as a global model into a direct, keyframe-based SLAM approach with an expectation-maximization (EM) framework. For tracking and map optimization, we propose an image alignment method that tracks the camera motion towards a reference keyframe and, at the same time, aligns the image with the planes in a global model. The model planes are concurrently extracted from the keyframes and estimated in global coordinates using graph optimization. By including both local frame-to-keyframe and global frame-to-model constraints in direct image alignment, we significantly reduce drift that is a typical problem of pure keyframe-based SLAM methods. Graph constraints between keyframes without overlapping views can be established if they observe the same model plane. An additional benefit of our method is that it provides a compact planar scene representation.

We name our algorithm CPA-SLAM and show a demo in Figure 5.1. On the first row we visualize for all keyframes their segmentation into planes and the association of these planes based on a minimal overlap criterion. It can be seen that there are some false segmentations, *e.g.,* small objects on and close to surfaces such as walls and tables are included into the plane segments. Shown one the second row, we illustrate that our EM tracking automatically determines not to align these pixels with the plane model. The final models on last two rows demonstrate the accurate mapping achieved with our method. It also shows the constraints between non-overlapping keyframes that are otherwise difficult to establish without the plane model. The major contributions of this work are summarized as follows:

- We develop an RGB-D SLAM approach that consistently tracks camera motion through direct image alignment towards a keyframe (as in **Kerl et al.** (2013a)) and a global plane model in an EM framework.

- We further use this alignment method to obtain spatial constraints between keyframes and the global plane model. These are jointly optimized with alignment constraints between keyframes for global consistency.

- We demonstrate a real-time capable SLAM system.

We compare our method to state-of-the-art approaches on challenging benchmark datasets and demonstrate improvements in trajectory estimation accuracy in relation to these approaches.

5.2 Related Work

Several approaches to visual SLAM have exploited planes as typical features in man-made environments. **Gee et al.** (2007) use planes to reduce the amount of interest points in a monocular Kalman-filter-based SLAM approach. In **Servant et al.** (2008), planes are also included in the state space of an EKF method to monocular SLAM. Their focus is on the incremental optimization of the planes in a global frame for augmented reality applications. In **Stückler and Behnke** (2008), orthogonality of planes in indoor environments is exploited to improve the consistency of plane models and SLAM. **Martinez-Carranza and Calway** (2010) propose a unified parameterization for points and planes in a monocular EKF-SLAM system.

In RGB-D SLAM, dense depth enables the detection of texureless planes. **Dou et al.** (2012) combine planes and interest points in frame-to-frame matching and bundle adjustment. They detect planes using a Hough-voting scheme. The corresponding planes between frames are found through RANSAC which yields plane tracks throughout the RGB-D sequence. Global planes are instantiated from the tracks and used within bundle adjustment. Since there may exist several tracks of the same plane entity, the planes are merged according to distance criteria. **Taguchi et al.** (2013) also use interest points and planes for SLAM with RGB-D sensors. They directly use various combinations of point and plane observations in a RANSAC framework to determine correspondence and camera pose between frames and a global map. **Trevor et al.** (2012) use RANSAC to find the major planes in a scene from RGB-D and 2D laser measurements. SLAM is performed in an EKF framework, associating plane observations with global planes in the map. **Salas-Moreno et al.** (2014) integrate incremental plane mapping into point-based fusion (**Keller et al.** 2013). Besides isolated surfels, they also label surfels to belong to the same planar structure and enforce planarity constraints in their estimates.

Some works also have tackled the problem of including object detection into SLAM. **Salas-Moreno et al.** (2013) detect objects from a model database and estimate their poses in individual frames. The poses of these objects in the global frame are then estimated through graph optimization. Another similar work is semantic bundle adjustment by **Fioraio and Stefano** (2013) in which the objects are detected and tracked, and included as 6-DoF landmarks in a bundle adjustment framework. In contrast to our approach, both methods do not handle a seamless transition between object and remaining image measurements already in the camera tracking part.

Our method applies direct alignment towards keyframe and plane model consistently in an

EM framework to estimate camera motion. We also use both keyframes and planes to obtain spatial constraints for global graph optimization.

5.3 Direct SLAM with Model Planes

Our SLAM system has a mixed nature of frame-to-keyframe and frame-to-model online tracking. As a global model, we maintain a set of planes that originate from keyframe segmentation to present the large smooth regions in the scene. We optimize the global plane model with all observations from the keyframes.

Figure 5.2 illustrates our SLAM system. At the front-end tracking, each RGB-D frame is aligned towards its nearest keyframe as well as the global plane model in an EM framework. An RGB-D frame becomes a new reference keyframe, if the certainty of motion estimation drops below a threshold. When a new keyframe is produced, an iteration of back-end optimization starts. In this process, we first segment the keyframe and associate the detected planes with the global model. Planes that are failed to be associated with existing planes in the model are included as new planes into the model. We further search for loop closures between keyframes and establish constraints between keyframe and the global model given the keyframe plane observations. The graph is then optimized to correct the keyframe poses and the plane model. The front-end tracking and back-end optimization can be processed in parallel.

In the following section, we will first explain the incremental construction of the plane model. We then detail our algorithm for camera tracking through direct image alignment towards a keyframe and the plane model. Last, we explain how we optimize the plane model and the camera poses through graph optimization.

5.3.1 Preliminaries

We denote the index of a keyframe by k and the current frame by i. An image is considered as a 2D continuous domain $\Omega \subset \mathbb{R}^2$ which can be segmented into disjoint regions Ω_j. We denote a 3D point by $\boldsymbol{v}$, its unit normal by $\mathbf{n}$, and a 2D pixel by $\mathbf{x}$. The projection of a 3D point onto the 2D image is $\mathbf{x} = \rho(\boldsymbol{v})$, and the back-projection is $\boldsymbol{v} = \rho^{-1}(\mathbf{x})$. To represent the rigid body motion in $\mathbb{SE}(3)$, we use the minimal parameterization with twist coordinate ξ of Lie algebra $\mathfrak{se}(3)$. The exponential map $g(\xi) = \exp(\widehat{\xi})$ converts the twist coordinate to a transformation matrix, and the log map g^{-1} operates vice versa. The subscript of ξ specifies direction: ξ_{ji} transforms from frame i to j, and $\xi_i = \xi_{wi}$ transforms from frame i to the world. The inverse motion is denoted by $\xi_i^{-1} = \xi_{iw}$. We further define function $t(\xi, \boldsymbol{v}) = g(\xi)\boldsymbol{v}$ to transform 3D points, and $\omega(\xi, \mathbf{x}) = \rho\left(t\left(\xi, \rho^{-1}(\mathbf{x})\right)\right)$ to warp pixels between frames.

We parametrize planes using the Hessian form $\boldsymbol{\pi} = (\mathbf{n}^\top, d)^\top$, where $\mathbf{n}$ is the unit plane normal, and $-d$ is the distance from the plane to the origin. The distance of any point $\boldsymbol{v}$ to the plane is calculated by $r_d = \mathbf{n}^\top \boldsymbol{v} + d$. A plane observed in frame k is transformed into

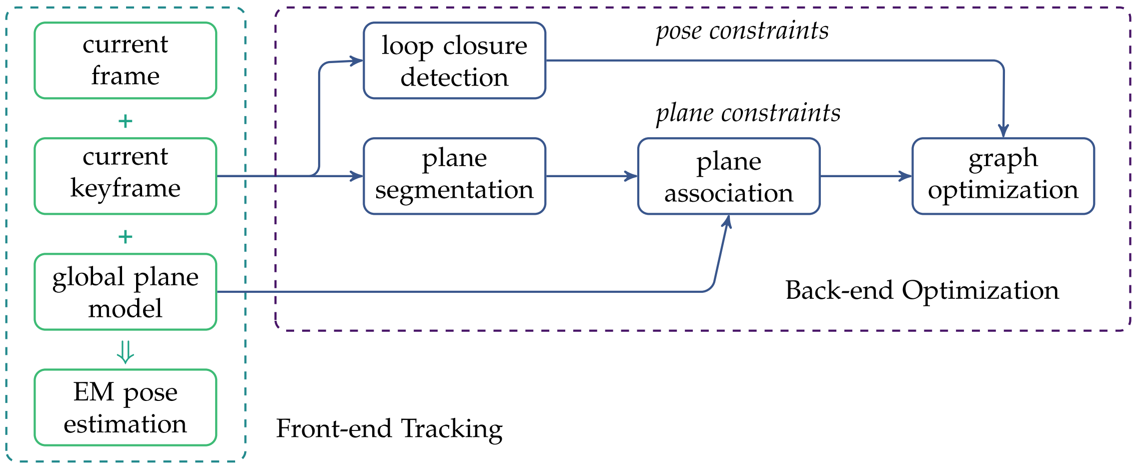


Figure 5.2: The schematic pipeline of the CPA SLAM system. At the front-end, the current frame is aligned to a keyframe and the global plane map in the EM motion estimation framework. A frame is classified as a keyframe if the tracking undertainty drops below threshold. Once a new keyframe is produced, the back-end optimization starts. Plane segmentation is performed and the detected segments are either associated with an existing plane, or is added as a new plane to the map. We then perform graph optimization to jointly estimate the optimal keyframe poses and the global plane map.

world coordinates by

$$t(\xi_k, \boldsymbol{\pi}) = g(\xi_k)^{-T} \boldsymbol{\pi} \,. \tag{5.1}$$

5.3.2 Global Plane Model

We define the global plane model as a set of planes in the world coordinates $\{\boldsymbol{\pi}_m^g\}$. This model represents large flat surfaces in a scene. Each plane $\boldsymbol{\pi}_m^g$ in the model is associated with a list of independent local observations $\boldsymbol{\pi}_{mk}$ in the keyframes. The global plane model is estimated through graph optimization using all local observations, which we detail in Section 5.3.6.

The global plane model is augmented incrementally. Whenever a new keyframe is produced, it is segmented into K regions, where Ω_0 is the non-planar region and $\Omega_j, j > 0$ is the jth plane. For efficient plane detection, we apply the agglomerative hierarchical clustering (AHC) algorithm proposed by **Feng et al.** (2014). For each plane segment, least squares plane fitting is used to estimate its parameters. We then associate the local observations with the global model. A correspondence is found if the angle between the plane normals is small ($< 15°$) and their distances to the origin are similar. We further confirm the association by warping the current plane segment into other keyframes and examine the overlaps. If a local plane observation fails to be associated, it is added to the global model as a new element.

5.3.3 Tracking towards Keyframe and Plane Model

We now describe motion estimation from the current frame i to the keyframe k by minimizing both photometric error r_I and geometric error r_G. To simplify the notation, we drop the subscript of ξ in this section.

5.3.3.1 Formulation

The photometric residual is defined on the intensity image assuming photoconsistency as

$$r_I = I_k\big(\omega(\xi, \mathbf{x})\big) - I_i(\mathbf{x}) \ . \tag{5.2}$$

The geometric residual is defined by

$$r_G = \begin{cases} \mathbf{n}_k^T\big(g(\xi, \mathbf{v}_i) - \mathbf{v}_k\big) & \text{if} \quad \omega(\xi, \mathbf{x}_i) \in \Omega_0 \\ \mathbf{n}_{\pi j}^T g(\xi, \mathbf{v}_i) + d_j & \text{if} \quad \omega(\xi, \mathbf{x}_i) \in \Omega_j \end{cases} , \tag{5.3}$$

which depends on whether the current pixel $\mathbf{x}_i$ is warped to the non-planar region Ω_0 or the jth planar region Ω_j of the keyframe. In case of Ω_0, the geometric residual resembles a variant of ICP (**Besl and McKay** 1992). Otherwise, the residual is the distance to the corresponding global plane transformed into the keyframe, thus, $(\mathbf{n}_{\pi j}^T, d_j)^T = t(\xi_k^{-1}, \pi_m^g)$.
Combining the photometric and geometric error into one variable $\mathbf{r} = (r_I, r_G)^T$, and with N correspondences between keyframe and current frame, we find the camera motion by minimizing the following non-linear weighted least squares,

$$\xi^* = \arg\min_{\xi} \sum_{n}^{N} \sum_{k}^{K} \gamma_{nk} w_{nk} \mathbf{r}_n^T \boldsymbol{\Sigma}_k^{-1} \mathbf{r}_n \ . \tag{5.4}$$

The weight w_{nk} is used to enhance the robustness against outliers and can be iteratively estimated. In our case, the weights are derived from a Student-t distribution as proposed in **Kerl et al.** (2013a). The variable $\boldsymbol{\gamma}_n \in \mathbb{R}^K$ is the labeling that indicates which region the residual belongs to. Accordingly, γ_{n0} refers to non-planar region Ω_0 and γ_{nj} refers planar region Ω_j. Instead of a hard labeling $\gamma_{nk} \in \{0, 1\}$, we use the soft labeling $\gamma_{nk} \in [0, 1]$ to increase robustness. This objective can be efficiently optimized with the Gauss-Newton method. In the following we will address how to concurrently determine $\boldsymbol{\gamma}, \boldsymbol{w}, \boldsymbol{\Sigma}$ in a probabilistic formulation.

5.3.3.2 A Probabilistic View on Motion Estimation

The optimization problem in Equation (5.4) cannot be solved directly, because the parameters $\boldsymbol{\gamma}, \boldsymbol{w}, \boldsymbol{\Sigma}$ also need to be estimated in addition to the motion ξ. To solve for both, we

More detailed analysis on t-distribution and EM estimation for mixture models are given in Section 3.1. The related direct tracking formulation and solutions to Jacobian computation is described in Section 3.2

motivate the same energy function from a probabilistic point of view and show that optimizing Equation (5.4) is equivalent to optimizing a mixture of bivariate t-distributions in an EM framework.

Suppose that $K-1$ planes are visible in the keyframe. For each pixel observation with residual $\mathbf{r}_n$, there is a corresponding indicator $\mathbf{z}_n \in \mathbb{B}^K$ that tells which segment the geometric observation comes from. As an indicator, $\mathbf{z}_n$ satisfies $z_{nk} \in \{0, 1\}$ and $\sum_k^K z_{nk} = 1$. Now, assume the following probability

$$p(z_{nk} = 1) = \eta_k \tag{5.5}$$

$$p(\mathbf{r}_n \mid z_{nk} = 1) = p_t(\mathbf{r}_n\,;\, 0, \boldsymbol{\Sigma}_k, \nu_k), \tag{5.6}$$

which can also be written in a compact form,

$$p(\mathbf{z}_n) = \prod_k^K \eta_k^{z_{nk}},\; p(\mathbf{r}_n \mid \mathbf{z}_n) = \prod_k^K p_t(\mathbf{r}_n\,;\, 0, \boldsymbol{\Sigma}_k, \nu_k)^{z_{nk}}. \tag{5.7}$$

The marginal probability of $\mathbf{r}_n$ is therefore,

$$p(\mathbf{r}_n) = \sum_k^K \eta_k p_t(\mathbf{r}_n\,;\, 0, \boldsymbol{\Sigma}_k, \nu_k), \tag{5.8}$$

which is a mixed model of bivariate t-distributions. Seeking motion ξ by maximum-likelihood estimation, yields

$$\xi^* = \arg\max_{\xi} \log p(\mathbf{r} \mid \xi) = \arg\max_{\xi} \sum_n^N \log\left(\sum_k^K \eta_k p_t(\mathbf{r}_n; 0, \boldsymbol{\Sigma}_k, \nu_k)\right).$$

With the distribution of $\mathbf{r}$ being a mixed model, the logarithm acts outside the summation and the direct optimization no longer yields a weighted least squares form. Even worse, there is no closed-form solution for the hyper-parameter $\boldsymbol{\Sigma}_k$.

The indicator $\mathbf{z}_n$ is a latent variable which we cannot observe directly. Therefore, the observed data $\mathbf{r}$ are incomplete, while $\{\mathbf{r}, \mathbf{z}\}$ are complete. Using Equation (5.7), the complete log-likelihood is,

$$\begin{aligned}\log p(\mathbf{r}, \mathbf{z}) &= \log \prod_n^N \prod_k^K \left(\eta_k p_t(\mathbf{r}_n\,;\, 0, \boldsymbol{\Sigma}_k, \nu_k)\right)^{z_{nk}} \\ &= \sum_n^N \sum_k^K z_{nk} \log\left(\eta_k p_t(\mathbf{r}_n\,;\, 0, \boldsymbol{\Sigma}_k, \nu_k)\right).\end{aligned} \tag{5.9}$$

This shows that with the knowledge of the indicators $\mathbf{z}$ we can regain the simple form of the optimization problems to estimate motion and hyper-parameters.

5.3.3.3 Tracking as Expectation-Maximization

The EM algorithm provides a probabilistic formalism to estimate the parameters of posterior probability functions with latent variables as in Equation (5.9). It constructs a lower bound on the log-likelihood by optimizing the Kullback-Leibler divergence between a simpler approximation and the actual posterior probability (see **Bishop** (2006) for details). In EM, one therefore optimizes the conditional expectation of the log joint probability, conditioned on the posterior probability of the latent variables. In our case, this is

$$\mathbb{E}_{p(\mathbf{z}|\mathbf{r})}\left[\log p(\mathbf{r},\mathbf{z})\right] = \sum_{n}^{N}\sum_{k}^{K} \gamma_{nk} \log\left(\eta_k p_t(\mathbf{r}_n\,;\,0,\boldsymbol{\Sigma}_k,\nu_k)\right), \tag{5.10}$$

where we define the mixing coefficient

$$\gamma_{nk} = \mathbb{E}_{p(\mathbf{z}|\mathbf{r})}[z_{nk}] = p(z_{nk} \mid \mathbf{r}_n). \tag{5.11}$$

Further writing out the EM objective function yields,

$$\mathbb{E}_{p(\mathbf{z}|\mathbf{r})}\left[\log p(\mathbf{r},\mathbf{z})\right] = \sum_{n}^{N}\sum_{k}^{K} \gamma_{nk} \cdot \frac{\nu_k + 2}{\nu_k + \mathbf{r}_n^T \boldsymbol{\Sigma}_k^{-1} \mathbf{r}_n} \cdot \mathbf{r}_n^T \boldsymbol{\Sigma}_k^{-1} \mathbf{r}_n. \tag{5.12}$$

Apparently, this corresponds to the previous objective function in Equation (5.4) by setting

$$w_{nk} = \frac{\nu_k + 2}{\nu_k + \mathbf{r}_n^T \boldsymbol{\Sigma}_k^{-1} \mathbf{r}_n}. \tag{5.13}$$

Now we can deduct the EM steps. In the $(t+1)$ E-step, we estimate the posterior probability of $\mathbf{z}$ holding parameters $\boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\Sigma}$ from the t M-step fixed. Working out the mathematical details, this yields

$$\gamma_{nk}^{t+1} = \frac{\eta_k^t p_t(\mathbf{r}_n; 0, \boldsymbol{\Sigma}_k^t, \nu_k)}{\sum_{j=1}^{K} \eta_j^t p_t\left(\mathbf{r}_n; 0, \boldsymbol{\Sigma}_j^t, \nu_k\right)}. \tag{5.14}$$

In the $(t+1)$ M-step, we in turn solve for the motion estimation holding the mixing coefficients from the $(t+1)$ E-step fixed. To this end, Equation (5.12) is iteratively linearized with first-order Taylor approximation of $\mathbf{r}$. At each iteration, this yields a normal equation to solve for an increment $\Delta\boldsymbol{\xi}$ on the motion,

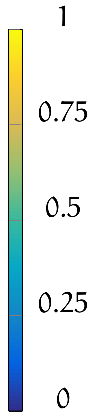
$$\sum_{n}^{N}\sum_{k}^{K} \gamma_{nk}^{t+1} w_{nk}^{t+1} \mathbf{J}_n^T \boldsymbol{\Sigma}_k^{-1} \mathbf{J}_n \Delta\xi = -\sum_{n}^{N}\sum_{k}^{K} \gamma_{nk}^{t+1} w_{nk}^{t+1} \mathbf{J}_n^T \boldsymbol{\Sigma}_k^{-1} \mathbf{r}_n. \tag{5.15}$$

The hyper-parameters are then updated by

$$\eta_k^{t+1} = \arg\max_{\eta_k} p_t(\mathbf{r}_n\,;\,0,\boldsymbol{\Sigma}_k,\nu_k)) = \frac{1}{N}\sum_{n}^{N} \gamma_{nk}^{t+1}\,, \tag{5.16}$$

$$\boldsymbol{\Sigma}_k^{t+1} = \arg\max_{\boldsymbol{\Sigma}_k} p_t(\mathbf{r}_n\,;\,0,\boldsymbol{\Sigma}_k,\nu_k)) = \frac{\sum_{n}^{N} \gamma_{nk}^{t+1} w_{nk}^{t+1} \mathbf{r}_n \mathbf{r}_n^T}{\sum_{n=1}^{N} \gamma_{nk}^{t+1}}. \tag{5.17}$$

(a) keyframe (b) keyframe plane segmentation



(c) hard labeling (d) EM, shown with $1-\gamma_{n0}$

Figure 5.3: Comparison between the hard labeling and EM soft labeling to associate planar points in the current frame. The soft labeling is more robust against the false segmentation in the keyframe, *e.g.*, the keyboard and the book are assigned 0 probability to being on the plane of the table.

(a)

(c)

(b)

(d)

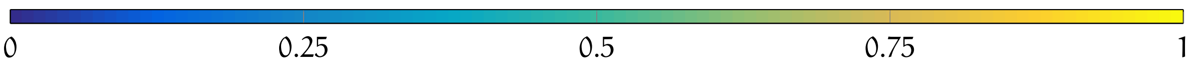


Figure 5.4: Properties of our tracking method: (a) keyframe; (b) plane segmentation; (c) the 1st, 11th, 21st, 31st and 44th (the last) frames that are registered to the keyframe; (d) corresponding EM soft labeling of (c), shown value of $1-\gamma_{n0}$. While EM trusts the keyframe data more when the current frame has a small temporal and spatial distance to the keyframe, it relies more on the global planar map otherwise.

From the above deduction, we see the alternating optimization principle of EM. The E-step computes the soft labeling given the current parameter values, while the M-step reestimates the parameters based on the latest soft labeling. To implement the EM framework, we use projective data association (**Newcombe et al.** 2011a) to guide the iterative EM steps. At each M-step, we warp the current frame into the keyframe and find correspondences for pixels if possible. With the segmented keyframe, the data association propagates the keyframe labeling to the current frame. Since we have small motion between frames and a good initial guess through tracking, the label propogation is mostly correct. Therefore, if a point is associated with plane j, we compute γ_{nj} according to Equation (5.14) and set $\gamma_{n0} = 1 - \gamma_{nj}$. Otherwise, we set $\gamma_{n0} = 1$. This implementation efficiently approximates the original EM solution due to the fact that a point most likely belongs to only one plane (up to plane intersections).

5.3.4 Properties of the EM Formulation

While our tracking method estimates camera motion, it also estimates the segmentation of the current frame via soft labeling. Note that a hard labeling can be easily obtained using projective data association. However, soft labeling is preferred, as it is more robust against false segmentation. We illustrate this in Figure 5.3, where the plane segmentation contains outliers (e.g., the keyboard and the book). These false detections are difficult to avoid due to noise and extreme scenarios with two close parallel planes.

Another property of our EM tracking is demonstrated in Figure 5.4, which shows the soft labeling of subsequent frames that are aligned to the same keyframe. In this example, tracking trusts the keyframe data more when the current frame has a small temporal and spatial distance to the keyframe, while it trusts the global plane model more when the current frame is further away. This is logical and as expected. With frames being close to the keyframe, the measurements very well correspond to keyframe data and, hence, direct alignment to the keyframe yields reliable and accurate registration. However, when a frame moves away from the keyframe both temporally and spatially, the difference between the measurements increases and drift accumulates. In such cases, the global plane model becomes beneficial.

5.3.5 Keyframe Selection and Loop Closure Detection

Following the work of **Kerl et al.** (2013a), keyframes are selected by examining the uncertainty of motion estimation. The normal equation (5.15) provides an approximate Hessian matrix $\mathbf{H}$,

$$\mathbf{H} = \sum_{n}^{N} \sum_{k}^{K} \gamma_{nk}^{t+1} w_{nk}^{t+1} J_n^T \boldsymbol{\Sigma}_k^{-1} J_n. \tag{5.18}$$

Its inverse gives a lower bound on the covariance of the estimated motion, i.e., $\boldsymbol{\Sigma}_{\xi} \approx \mathbf{H}^{-1}$. Assuming $\xi \sim \mathcal{N}(\xi^*, \boldsymbol{\Sigma}_{\xi})$, we can extract the uncertainty embedded in covariance into a scalar value using the differential entropy $h(\xi) = 3\big(1 + \ln(2\pi)\big) + 0.5 \ln(|\boldsymbol{\Sigma}_{\xi}|)$. Given the current keyframe k, we test the entropy ratio for every consecutive frame tracked towards the keyframe by $\alpha = h(\xi_{k+j})/h(\xi_{k+1})$. Whenever α drops below a pre-defined threshold, the $(k+j)$th frame is selected as the new keyframe. Empirically, we find the value range $0.9 \sim 0.95$ to generate good performance.

Whenever a new keyframe is produced, we find loop closures by comparing the current keyframe to previous keyframes via a spatial search. To register two keyframes, we use direct image alignment and initialize the estimation with the transformation computed from their poses. The same ratio test is performed to determine a successful closure. After the last keyframe being produced, we run an additional loop closure search for all keyframes.

5.3.6 Joint Pose and Plane Graph Optimization

On the global scale, we optimize the keyframe poses and the model planes for consistency in a graph

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \sum_{i,j} \mathbf{e}_{ij}^T \mathbf{H}_{ij} \mathbf{e}_{ij}, \tag{5.19}$$

where $\mathbf{e}_{ij}$ is the error of the edge connecting vertices i, j and $\mathbf{H}_{ij}$ is the information matrix. The variable $\boldsymbol{\Theta}$ is the list of the parameters to be optimized. In our case, $\boldsymbol{\Theta} = (\xi_1, \xi_2, \dots, \xi_N, \boldsymbol{\pi}_1^g, \boldsymbol{\pi}_2^g, \dots, \boldsymbol{\pi}_M^g)$. Since there are two types of vertices, keyframe poses and global planes, the graph also consists of two types of edges: between two poses and between a plane and a keyframe pose.

For an edge connecting two poses ξ_i and ξ_j with the measured constraint ξ_{ji}, the edge error is calculated by

$$\mathbf{e}_{ij} = g^{-1}\left(g(\xi_i^{-1})g(\xi_j)g(\xi_{ji})\right), \tag{5.20}$$

The Hessian in Equation (5.18) is used as the information matrix.

Now we define the error for an edge connecting a global plane $\boldsymbol{\pi}_j^g$ and a keyframe pose ξ_i, given the local plane observation $\boldsymbol{\pi}_{ji}$ in the keyframe. Notice that the Hessian plane equation is an over-parameterization of 3D planes, since a plane has only three degrees of freedom. Therefore, the Hessian form will lead to complications in the optimization, which requires extra constraints to ensure the unit length of the plane normal. To avoid this problem, we use the minimal parameterization $\boldsymbol{\tau} = (\phi, \psi, d)$, where ϕ and ψ are the azimuth and elevation angle of the normal, respectively. The following conversion between the Hessian and the minimal representation applies,

$$\boldsymbol{\tau} = q(\boldsymbol{\pi}) = \left(\phi = \arctan\frac{n_y}{n_x}, \psi = \arccos n_z, d\right)^T, \tag{5.21}$$

$$q^{-1}(\boldsymbol{\tau}) = \left(\cos\phi\cos\psi, \sin\phi\cos\psi, -\sin\psi, d\right)^T. \tag{5.22}$$

To avoid the singularities of the minimal representation, we force the angle to fall into $(-\pi, \pi]$. The error for plane-keyframe edges is then defined as

$$\mathbf{e}_{ij} = q\left(\boldsymbol{\pi}_j^g\right) - q\left(t(\xi_i, \boldsymbol{\pi}_{ji})\right). \tag{5.23}$$

Figure 5.5: Comparison of accumulated point cloud from keyframes, left to right: tracking without planes, hard labeling the planes, and soft labeling the planes.

The information matrix for plane-keyframe edges is set to isotropic using $\sigma_\pi^{-2}I$. We optimize the graph when a new keyframe is inserted into the graph using the g2o framework by **Kümmerle et al.** (2011).

5.4 Experimental Results

In this section, we evaluate our EM tracking method and the overall SLAM algorithm. Two public datasets with ground-truth trajectories are used in the assessment: the TUM RGB-D benchmark (**Sturm et al.** 2012) and the ICL-NUIM synthetic scenes (**Handa et al.** 2014). We also evaluate our algorithm with the Stanford scene3D dataset by **Zhou and Koltun** (2013), which has longer and more complicated trajectories. The EM tracking is implemented with CUDA and run on an NVidia GTX780 GPU with 2304 cores, 3.7 GHz, and 3 GB memory. The remaining SLAM methods are implemented in C++ to run on CPU and evaluated with an Intel Core i7-2660, 3.4 GHz and 8 GB RAM.

Table 5.1: The RMSE of absolute trajectory error (no final optimization) of frame-to-keyframe tracking methods: without plane model, plane model with a hard labeling and plane model with soft labeling (bold marks the best).

dataset	without plane	hard labeling	soft labeling
fr1/desk	0.034	0.080	**0.030**
fr1/plant	**0.050**	0.072	0.073
fr2/desk	0.097	0.134	**0.095**
fr3/office	0.086	0.077	**0.076**
fr3/structure_texture_near	0.049	**0.028**	0.036
fr3/nst	0.076	**0.032**	**0.032**
iclnuim/lr3	**0.002**	0.049	**0.002**
iclnuim/lr3noisy	0.028	0.024	**0.019**

In the first experiment, we evaluate the performance of our EM tracking method by comparing with two other variants: a) tracking without planes and b) tracking with planes using hard-labeling. We set $\alpha = 0.9$ for keyframe selection in all methods and compare the

Table 5.2: Comparison of our CPA SLAM to other SLAM algorithms using planes. The RMSE of the absolute trajectory error (m) are shown and the results of other methods are cited from the original papers (bold marks the best).

dataset	CPA SLAM	planar SLAM [160]	point-plane SLAM [182]
iclnuim/lr0noisy	**0.007**	0.246	–
iclnuim/lr1noisy	**0.006**	0.017	–
fr1/xyz	**0.011**	–	0.024
fr1/floor	0.085	–	**0.065**

Table 5.3: The RMSE of the absolute trajectory error (m) of CPA SLAM, in comparison to state-of-the-art algorithms: DVO SLAM (**Kerl et al.** 2013a), Kintinuous with deformable mapping (**Whelan et al.** 2015b), MRSMap (**Stückler and Behnke** 2012), and RGB-D SLAM (**Endres et al.** 2012). The best results are marked in bold.

dataset	CPA SLAM	DVO SLAM	Kintinous	MRSMap	RGB-D SLAM
fr1/desk	**0.018**	0.021	0.037	0.043	0.023
fr1/desk2	**0.029**	0.046	0.071	0.049	0.043
fr1/plant	0.029	0.028	0.047	**0.026**	0.091
fr1/room	0.055	**0.053**	0.075	0.069	0.084
fr1/rpy	0.024	**0.020**	0.028	0.027	0.026
fr1/xyz	**0.011**	**0.011**	0.017	0.013	0.014
fr2/desk	0.046	**0.017**	0.034	0.052	0.095
fr2/xyz	**0.014**	0.018	0.029	0.020	0.026
fr3/office	**0.025**	0.035	0.030	0.042	–
fr3/nst	**0.016**	0.038	0.031	1.530	–
iclnuim/lr2noisy	**0.089**	0.339	0.129	0.331	–
iclnuim/lr3noisy	**0.009**	0.152	0.864	1.127	–

obtained trajectories without the final graph optimization. Table 5.1 shows the root mean square error (RMSE) of the absolute trajectory error (ATE) and Figure 5.5 gives a visual comparison of the alignment results. It can be seen that our tracking improves trajectory accuracy and yields better consistent models.

In the second experiment, we evaluate the benefits of the global plane model in our full CPA-SLAM system. Table 5.2 presents a comparison between our method and two other SLAM algorithms that use planes: dense planar SLAM by **Salas-Moreno et al.** (2014) and point-plane SLAM by **Taguchi et al.** (2013). Our EM-based algorithm achieves much better accuracy on the sequences. These improvements come from joining the advantages of keyframe-based and model-based tracking into a flexible and robust EM framework.

We also compare our CPA-SLAM algorithm to several state-of-the-art RGB-D SLAM systems,

Figure 5.6: Fused model from our SLAM methods. The trajectories with and without graph optimization are shown in blue and red, and the constraints between pose and planes are shown in gray.

Table 5.4: The average runtime performance of our algorithm, measured in millisecond. Frame tracking runs well within the typical camera frame rate of 30fps. Plane segmentation and association has to run only for each keyframe. The comparably slow graph optimization runs in a background thread. Therefore, our RGB-D SLAM is able to perform in realtime.

	fr3/office	fr3/nst	iclnuim/lr3noisy
keyframes/ frames	118/ 2489	66/ 1637	59/ 1241
global planes/ local planes	18/ 197	1/ 66	7/ 64
frame tracking (ms)	20.9	24.1	23.2
plane segmentation (ms)	9.3	14.4	9.8
plane association (ms)	10.7	8.4	8.4
graph optimization (ms)	75.8	24.0	22.0

including DVO-SLAM from **Kerl et al.** (2013a), Kintinuous with deformable mapping from **Whelan et al.** (2015b), MRSMap from **Stückler and Behnke** (2012), and RGB-D SLAM from **Endres et al.** (2012). We report the ATE of the final trajectories in Table 5.2. The results showcase that our SLAM algorithm performs better or on par with the state-of-the-art algorithms on the sequences. For sequences with many planar structures or containing heavy noise, *e.g.*, fr3/nst, iclnuim/lr2noisy and iclnuim/lr3noisy, our algorithm demonstrates most advantages in reducing tracking errors. Figure 5.6 shows the output of our SLAM methods. It can be seen that with a global plane model, constraints between non-overlapping keyframes are established, as long as they observe the same global plane. As can be seen, geometry is accurately reconstructed from the estimated trajectory.

Table 5.4 presents the runtime of the computationally intensive parts of our method. Tracking achieves real-time performance with approximately 50 fps. Plane segmentation is also efficiently performed with approximately 100 fps. Combining frame tracking, plane segmentation and association, our implementation requires around 40 ms to process one keyframe. Searching loop closures and optimizing the graph requires relatively long time, however, this can be run on parallel CPU threads. As a result, our method is capable of real-time performance for 30 fps RGB-D mapping.

5.5 Conclusion

In this paper, we proposed a novel method that combines direct image alignment and global model alignment for RGB-D SLAM. Our method tracks camera motion towards the nearest keyframe and the global plane model in an EM framework. This reduces drift and establishes constraints among non-overlapping keyframes that observe the same plane. The keyframe poses and the plane model are optimized in one graph concurrently. Our method exhibits state-of-the-art accuracy on publicly available benchmark datasets and is capable of real-time performance. In future work, we will consider the integration of further types of geometric shapes and complex objects into our SLAM system. One important question here is how to learn and acquire object models on-the-fly, and how to come to adequate object hypotheses.

CHAPTER 6

FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture

ABSTRACT With the availability of RGB-D cameras, it is expected that additional depth measurement will improve the accuracy. Here we investigate a solution how to incorporate complementary depth information into a semantic segmentation framework by making use of convolutional neural networks (CNNs). Recently encoder-decoder type fully convolutional CNN architectures have achieved a great success in the field of semantic segmentation. Motivated by this observation we propose an encoder-decoder type network, where the encoder part is composed of two branches of networks that simultaneously extract features from RGB and depth images and fuse depth features into the RGB feature maps as the network goes deeper. Comprehensive experimental evaluations demonstrate that the proposed fusion-based architecture achieves competitive results with the state-of-the-art methods on the challenging SUN RGB-D benchmark obtaining 76.27% global accuracy, 48.30% average class accuracy and 37.29% average intersection-over-union score.

6.1 Introduction

Visual scene understanding in a glance is one of the most amazing capability of the human brain. In order to model this ability, semantic segmentation aims at giving a class label for each pixel on the image according to its semantic meaning. This problem is one of the most challenging tasks in computer vision, and has received a lot of attention from the computer

© Springer. Reprinted, with permission, from Caner Hazirbas, Lingni Ma, Csaba Domokos and Daniel Cremers, *FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture*, In proc. of Asian Conference on Computer Vision (ACCV), Nov 2016. Revised layout, minor adaptations.

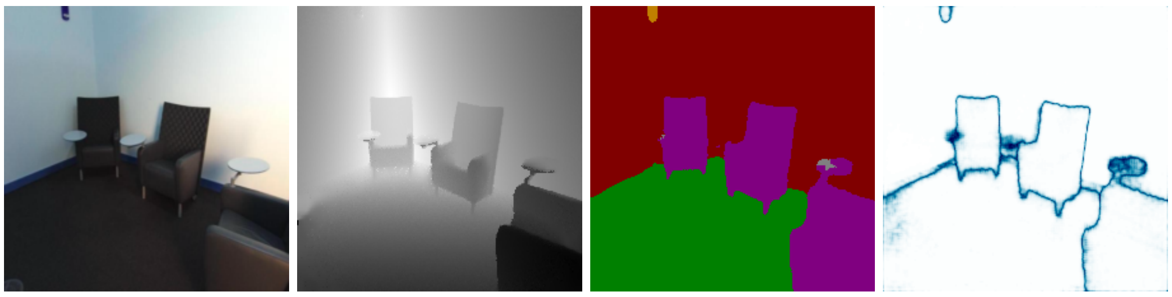


Figure 6.1: An exemplar output of FuseNet. From left to right: input RGB and depth images, the predicted semantic labeling and the probability of the corresponding labels, where white and blue denote high and low probability, respectively.

vision community (**Byeon et al.** 2015; **Gupta et al.** 2014; **G. Lin et al.** 2016b; **Long et al.** 2015; **Noh et al.** 2015; **Pinheiro and Collobert** 2014; **Zheng et al.** 2015).

Convolutional neural networks (CNNs) have recently attained a breakthrough in various classification tasks such as semantic segmentation. CNNs have been shown to be powerful visual models that yields hierarchies of features. The key success of this model mainly lies in its general modeling ability for complex visual scenes. Currently CNN-based approaches (**L.-C. Chen et al.** 2015; **Long et al.** 2015; **Zheng et al.** 2015) provide the state-of-the-art performance in several semantic segmentation benchmarks. In contrast to CNN models, by applying hand-crafted features one can generally achieve rather limited accuracy.

Utilizing depth additional to the appearance information (*i.e.*, RGB) could potentially improve the performance of semantic segmentation, since the depth channel has complementary information to RGB channels, and encodes structural information of the scene. The depth channel can be easily captured with low cost RGB-D sensors. In general object classes can be recognized based on their color and texture attributes. However, the auxiliary depth may reduce the uncertainty of the segmentation of objects having similar appearance information. **Couprie et al.** (2013) observed that the segmentation of classes having similar depth, appearance and location is improved by making use of the depth information too, but it is better to use only RGB information to recognize object classes containing high variability of their depth values. Therefore, the optimal way to fuse RGB and depth information has been left an open question.

In this paper we address the problem of indoor scene understanding assuming that both RGB and depth information simultaneously available (see Figure 6.1). This problem is rather crucial in many perceptual applications including robotics. We remark that although indoor scenes have rich semantic information, they are generally more challenging than outdoor scenes due to more severe occlusions of objects and cluttered background. For example, indoor object classes, such as *chair*, *dining table* and *curtain* are much harder to recognize than outdoor classes, such as *car*, *road*, *building* and *sky*. The contribution of the paper can be summarized as follows:

- We investigate a solution how to incorporate complementary depth information into a semantic segmentation framework. For this sake we propose an encoder-decoder type network, referred to as FuseNet, where the encoder part is composed of two branches of networks that simultaneously extract features from RGB and depth images and fuse depth features into the RGB feature maps as the network goes deeper (see Figure 6.2).

- We propose and examine two different ways for fusion of the RGB and depth channels. We also analyze the proposed network architectures, referred to as dense and sparse fusion (see Figure 6.3), in terms of the level of fusion.

- We experimentally show that our proposed method is successfully able to fuse RGB and depth information for semantic segmentation also on cluttered indoor scenes. Moreover, our method achieves competitive results with state-of-the-art methods in terms of segmentation accuracy evaluated on the challenging SUN RGB-D dataset (**Song et al.** 2015).

6.2 Related Work

A fully convolutional network (FCN) architecture has been introduced in **Long et al.** (2015) that combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce accurate and detailed segmentations by applying end-to-end training. **Noh et al.** (2015) have proposed a novel network architecture for semantic segmentation, referred to as DeconvNet, which alleviates the limitations of fully convolutional models (*e.g.,* , very limited resolution of labeling). DeconvNet is composed of deconvolution and unpooling layers on top of the 16-layer VGGNet (**Simonyan and Zisserman** 2015). To retrieve semantic labeling on the full image size, **M. D. Zeiler and Fergus** (2014) have introduced a network composed of deconvolution and unpooling layers. Concurrently, a very similar network architecture has been presented in **Badrinarayanan et al.** (2017) based on the 16-layer VGGNet (**Simonyan and Zisserman** 2015), referred to as SegNet. In contrast to DeconvNet, SegNet consists of smoothed unpooled feature maps with convolutions instead of deconvolutions. **Kendall et al.** (2015) further improved the segmentation accuracy of SegNet by applying dropout (**Srivastava et al.** 2014) during training and inference (**Gal and Ghahramani** 2016).

Some recent semantic segmentation algorithms combine the strengths of CNN and conditional random field (CRF) models. It has been shown that the poor pixel classification accuracy, due to the invariance properties that make CNNs good for high level tasks, can be overcome by combining the responses of the CNN at the final layer with a fully connected CRF model (**L.-C. Chen et al.** 2015). CNN and CRF models have also been combined in **Zheng et al.** (2015). More precisely, the method proposed in **Zheng et al.** (2015) applies mean field approximation as the inference for a CRF model with Gaussian pairwise potentials, where the mean field approximation is modeled as a recurrent neural network, and the defined network is trained end-to-end refining the weights of the CNN model. Recently, **G. Lin et al.** (2016a); **G. Lin et al.** (2016b) have also combined CNN and CRF models for learning patch-patch context between image regions, and have achieved the current state-of-the-art performance in semantic segmentation. One of the main ideas in **G. Lin et al.** (2016a) is to define CNN-based pairwise potential functions to capture semantic correlations between neighboring patches. Moreover, efficient piecewise training is applied for the CRF model in order to avoid repeated expensive CRF inference during the course of back-propagation.

In **Pinheiro and Collobert** (2014) a feed-forward neural network has been proposed for scene labeling. The long range (pixel) label dependencies can be taken into account by capturing sufficiently large input context patch, around each pixel to be labeled. The method by **Pinheiro and Collobert** (2014) relies on a recurrent convolutional neural networks (RCNN), *i.e.,* a sequential series of networks sharing the same set of parameters. Each instance takes as input both an RGB image and the predictions of the previous instance of the network. RCNN-based approaches are known to be difficult to train, in particular, with large data, since long-term dependencies are vanished while the information is accumulated by the recurrence (**Byeon et al.** 2015).

Byeon et al. (2015) have presented long short term memory (LSTM) recurrent neural networks for natural scene images taking into account the complex spatial dependencies of labels. LSTM networks have been commonly used for sequence classification. These networks include recurrently connected layers to learn the dependencies between two frames, and then transfer the probabilistic inference to the next frame. This allows to easily memorize the context information for long periods of time in sequence data. It has been shown in **Byeon et al.** (2015) that LSTM networks can be generalized well to any vision-based task and efficiently capture local and global contextual information with a low computational complexity.

State-of-the-art CNNs have the ability to perform segmentation on different kinds of input sources such as RGB or even RGB-D. Therefore a trivial way to incorporate depth information would be to stack it to the RGB channels and train the network on RGB-D data assuming a four-channel input. However, it would not fully exploit the structure of the scene encoded by the depth channel. This will be also shown experimentally in Section 6.4. By making use of deeper and wider network architecture one can expect the increase of the robustness and the accuracy. Hence, one may define a network architecture with more layers. Nevertheless, this approach would require huge dataset in order to learn all the parameter making the training infeasible even in the case when the parameters are initialized with a pre-trained network.

6.2.1 The State of the Arts on RGB-D Data

A new representation of the depth information has been presented by **Gupta et al.** (2014). This representation, referred to as HHA, consists of three channels: disparity, height of the pixels and the angle between of normals and the gravity vector based on the estimated ground floor, respectively. By making use of the HHA representation, a superficial improve-

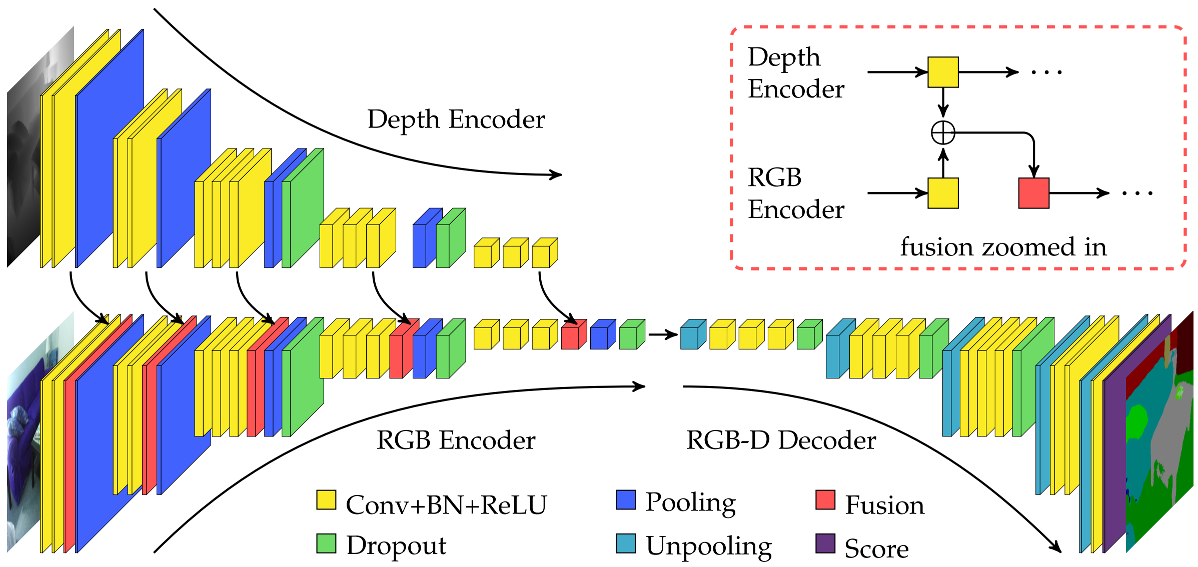


Figure 6.2: The architecture of FuseNet proposed in this paper, where the layers are color coded. The network contains two branches to extract features from the RGB and the depth input, and the feature maps from depth is constantly accvfused into the RGB branch, denoted with the red arrows. In our architecture, fusion layer is implemented as element-wise summation, as detailed in the dashed box.

ment was achieved in terms of segmentation accuracy (**Gupta et al.** 2014). On the other hand, the information retrieved only from the RGB channels still dominates the HHA representation. As we shall see in Section 6.4, the HHA representation does not hold more information than the depth itself. Furthermore, computing HHA representation requires high computational cost. In this paper we investigate a better way of exploiting depth information with less computational burden.

Z. Li et al. (2016b) have introduced a novel LSTM Fusion (LSTM-F) model that captures and fuses contextual information from photometric and depth channels by stacking several convolutional layers and an LSTM layer. The memory layer encodes both short- and long-range spatial dependencies in an image along vertical direction. Moreover, another LSTM-F layer integrates the contexts from different channels and performs bi-directional propagation of the fused vertical contexts. In general, these kinds of architectures are rather complicated and hence more difficult to train. In contrast to recurrent networks, we propose a simpler network architecture.

6.3 FuseNet: Unified CNN Framework for Fusing RGB and Depth Channels

We aim to solve the semantic segmentation problem on RGB-D images. We define the label set as $\mathcal{L} = \{1, 2, \dots, K\}$. We assume that we are given a training set

$$\mathcal{D} = \left\{ (\mathbf{X}_i, \mathbf{Y}_i) \mid \mathbf{X}_i \in \mathbb{R}^{H \times W \times 4}, \mathbf{Y}_i \in \mathcal{L}^{H \times W} \;\; \forall i \in \{1, \dots, M\} \right\}$$

consisting of M four-channel RGB-D images ($\mathbf{X}_i$), having the same size $H \times W$, along with the ground-truth labeling ($\mathbf{Y}_i$). Moreover, we assume that the pixels are drawn as *i.i.d.* samples following a categorical distribution. Based on this assumption, we may define a CNN model to perform multinomial logistic regression.

The network extracts features from the input layer and through filtering provides classification score for each label as an output at each pixel. We model the network as a composition of functions corresponding to L layers with parameters denoted by $\boldsymbol{W} = \left(\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \dots, \boldsymbol{w}^{(L)}\right)^{\mathsf{T}}$, that is

$$f(\mathbf{x}; \boldsymbol{W}) = g^{(L)}(g^{(L-1)}\left(\cdots g^{(2)}\left(g^{(1)}\left(\mathbf{x}; \boldsymbol{w}^{(1)}\right); \boldsymbol{w}^{(2)}\right) \cdots ; \boldsymbol{w}^{(L-1)}); \boldsymbol{w}^{(L)}\right) . \tag{6.1}$$

The classification score of a pixel $\mathbf{x}$ for a given class c is obtained from the function $f_c(\mathbf{x}; \boldsymbol{W})$, which is the cth component of $f(\mathbf{x}; \boldsymbol{W})$. Using the *softmax* function, we can map this score to a probability distribution

$$p(c \mid \mathbf{x}, \boldsymbol{W}) = \frac{\exp\left(f_c(\mathbf{x}; \boldsymbol{W})\right)}{\sum_{k=1}^{K} \exp\left(f_k(\mathbf{x}; \boldsymbol{W})\right)} . \tag{6.2}$$

For the training of the network, *i.e.,* learning the optimal parameters $\boldsymbol{W}^*$, the cross-entropy loss is used, which minimizes the KL-divergence between the predicted and the true class distribution:

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \frac{1}{2}\|\boldsymbol{W}\|^2 - \frac{\lambda}{MHW} \sum_{i=1}^{M} \sum_{j=1}^{HW} \log p\left(y_{ij} \mid \mathbf{x}_{ij}, \boldsymbol{W}\right) , \tag{6.3}$$

where $\mathbf{x}_{ij} \in \mathbb{R}^4$ stands for the jth pixel of the ith training image and $y_{ij} \in \mathcal{L}$ is its ground-truth label. The hyper-parameter $\lambda > 0$ is chosen to apply weighting for the regularization of the parameters (*i.e.,* ℓ_2-norm of $\boldsymbol{W}$).

At inference, a probability distribution is predicted for each pixel via softmax normalization, defined in Equation (6.2), and the labeling is calculated based on the highest class probability.

6.3.1 FuseNet Architecture

We propose an encoder-decoder type network architecture as shown in Figure 6.2. The proposed network has two major parts: 1) the *encoder* part extracts features and 2) the *decoder* part upsamples the feature maps back to the original input resolution. This encoder-decoder style has been already introduced in several previous works such as DeconvNet (**Noh et al.** 2015) and SegNet (**Badrinarayanan et al.** 2017; **Kendall et al.** 2015) and has achieved good segmentation performance. Although our proposed network is based on this type of architecture, we further consider to have two encoder branches. These two branches extract features from RGB and depth images. We note that the depth image is normalized to have the same value range as color images, *i.e.,* into the interval of [0,255]. In order to combine

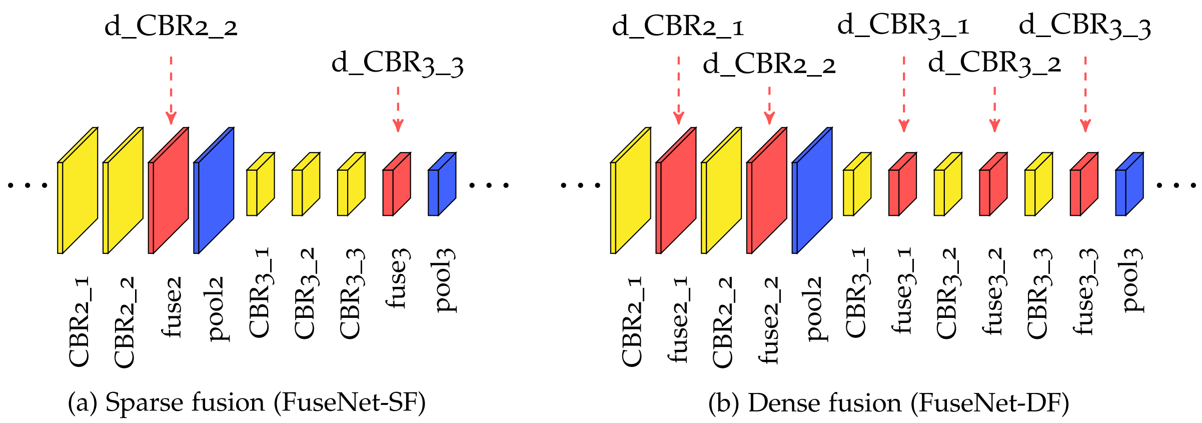


Figure 6.3: Illustration of different fusion strategies at the second (CBR2) and third (CBR3) convolution blocks of VGG 16-layer net. (a) The fusion layer is only inserted before each pooling layer. (b) The fusion layer is inserted after each CBR block.

information from both input modules, we fuse the feature maps from the depth branch into the feature maps of the RGB branch. We refer to this architecture as *FuseNet* (see Figure 6.2).

The encoder part of FuseNet resembles the 16-layer VGG net (**Simonyan and Zisserman** 2015), except of the fully connected layers FC6, FC7 and FC8, since the fully connected layers reduce the resolution with a factor of 49, which increases the difficulty of the upsampling part. In our network, we always use batch normalization (BN) after convolution (Conv) and before rectified linear unit[1] (ReLU) to reduce the internal covariate shift (**Ioffe and Szegedy** 2015). We refer to the combination of convolution, batch normalization and ReLU as CBR block, respectively. The BN layer first normalizes the feature maps to have zero-mean and unit-variance, and then scales and shifts them afterwards. In particular, the scale and shift parameters are learned during training. As a result, color features are not overwritten by depth features, but the network learns how to combine them in an optimal way.

The decoder part is a counterpart of the encoder part, where memorized unpooling is applied to upsample the feature maps. In the decoder part, we again use the CBR blocks. We also did experiments with deconvolution instead of convolution, and observed very similar performance. As proposed in **Kendall et al.** (2015), we also apply dropout in both the encoder and the decoder parts to further boost the performance. However, we do not use dropout during test time.

The key ingredient of the FuseNet architecture is the fusion block, which combines the feature maps of the depth branch and the RGB branch. The fusion layer is implemented as element-wise summation. In FuseNet, we always insert the fusion layer after the CBR block.

[1] The rectified linear unit is defined as $\sigma(x) = \max(0, x)$.

By making use of fusion the discontinuities of the features maps computed on the depth image are added into the RGB branch in order to enhance the RGB feature maps. As it can be observed in many cases, the features in the color domain and in the geometric domain complement each other. Based on this observation, we propose two fusion strategies: a) dense fusion (FuseNet-DF), where the fusion layer is added after each CBR block of the RGB branch. b) sparse fusion (FuseNet-SF), where the fusion layer is only inserted before each pooling. These two strategies are illustrated in Figure 6.3.

6.3.2 Fusion of Feature Maps

In this section, we reason the fusion of the feature maps between the RGB and the depth branches. To utilize depth information a simple way would be just stacking the RGB and depth images into a four-channel input. However, we argue that by fusing RGB and depth information the feature maps are usually more discriminant than the ones obtained from the stacked input.

As we introduced before in Equation (6.1), each layer is modeled as a function g that maps a set of input $\mathbf{x}$ to a set of output $\mathbf{a}$ with parameter $\mathbf{w}$. We denote the kth feature map in the lth layer by $g_k^{(l)}$. Suppose that the given layer operation consists of convolution and ReLU, therefore

$$\mathbf{x}_k^{(l+1)} = g_k^{(l)}\left(\mathbf{x}^{(l)}; \mathbf{w}_k^{(l)}\right) = \sigma\left(\langle \mathbf{w}_k^{(l)}, \mathbf{x}^{(l)} \rangle + b_k^{(l)}\right) .$$

If the input is a four-channel RGB-D image, then the feature maps can be decomposed as $\mathbf{x} = \left(\mathbf{a}^\top, \mathbf{b}^\top\right)^\top$, where $\mathbf{a} \in \mathbb{R}^{d_1}$, $\mathbf{b} \in \mathbb{R}^{d_2}$ with $\dim(\mathbf{x}) := d = d_1 + d_2$ are features learned from the color channels and from the depth channel, respectively. According to this observation, we may write that

$$\begin{aligned}
\mathbf{x}_k^{(l+1)} &= \sigma\left(\langle \mathbf{w}_k^{(l)}, \mathbf{x}^{(l)} \rangle + b_k^{(l)}\right) = \sigma\left(\langle \mathbf{u}_k^{(l)}, \mathbf{a}^{(l)} \rangle + c_k^{(l)} + \langle \mathbf{v}_k^{(l)}, \mathbf{b}^{(l)} \rangle + d_k^{(l)}\right) \\
&= \max\left(0, \langle \mathbf{u}_k^{(l)}, \mathbf{a}^{(l)} \rangle + c_k^{(l)} + \langle \mathbf{v}_k^{(l)}, \mathbf{b}^{(l)} \rangle + d_k^{(l)}\right) \\
&\leqslant \max(0, \langle \mathbf{u}_k^{(l)}, \mathbf{a}^{(l)} \rangle + c_k^{(l)}) + \max\left(0, \langle \mathbf{v}_k^{(l)}, \mathbf{b}^{(l)} \rangle + d_k^{(l)}\right) \\
&= \sigma\left(\langle \mathbf{u}_k^{(l)}, \mathbf{a}^{(l)} \rangle + c_k^{(l)}\right) + \sigma\left(\langle \mathbf{v}_k^{(l)}, \mathbf{b}^{(l)} \rangle + d_k^{(l)}\right) ,
\end{aligned} \tag{6.4}$$

where we applied the decomposition of $\mathbf{w}_k^{(l)} = \left(\mathbf{u}_k^{(l)^\top}, \mathbf{v}_k^{(l)^\top}\right)^\top$ and $b_k^{(l)} = c_k^{(l)} + d_k^{(l)}$.

Based on the inequality in Equation (6.4), we show that the fusion of activations of the color and the depth branches (*i.e.*, their element-wise summation) produces a stronger signal than the activation on the fused features. Nevertheless, the stronger activation does not necessarily lead to a better accuracy. However, with fusion, we do not only increase the neuron-wise activation values, but also preserve activations at different neuron locations. The intuition behind this can be seen by considering low-level features (*e.g.*, edges). Namely, due to the fact that the edges extracted in RGB and depth images are usually complementary to each other. One may combine the edges from both inputs to obtain more information.

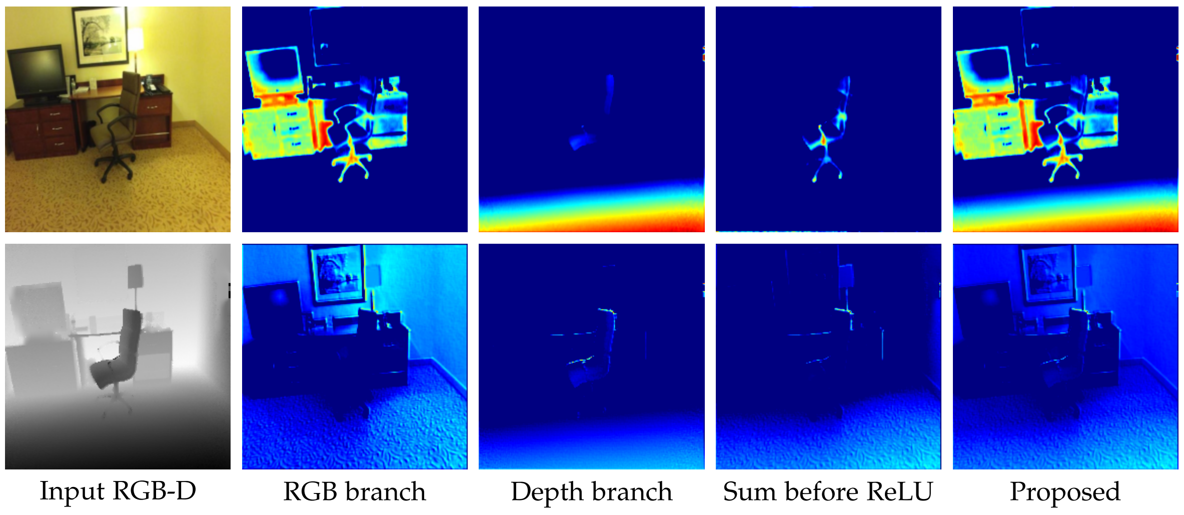


Figure 6.4: Comparison of two out of 64 feature maps produced at the CBR1_1 layer. The features from RGB and depth mostly compensate each other, where the textureless region usually have rich structure features and structureless regions usually present texture features. This visually illustrates that the proposed fusion strategy better preserves the informative features from color and depth than applying element-wise summation followed by ReLU.

Consequently, these low-level features help the network to extract better high-level features, and thus enhance the ultimate accuracy.

To demonstrate the advantage of the proposed fusion, we visualize the feature maps produced by CBR1_1 in Figure 6.4, which corresponds to low-level feature extraction (*e.g.*, edges). As it can be seen the low-level features in RGB and depth are usually complementary to each other. For example, the textureless region can be distinguished by its structure, such as the lap against the wall, whereas the structureless region can be distinguished by the color, such as the painting on the wall. While combining the feature maps before the ReLU layer fail to preserve activations, however, the proposed fusion strategy, applied after the ReLU layer, preserves well all the useful information from both branches. Since low-level features help the network to extract better high-level ones, the proposed fusion thus enhances the ultimate accuracy.

6.4 Experimental Evaluation

In this section, we evaluate the proposed network through extensive experiments. For this purpose, we use the publicly available SUN RGB-D scene understanding benchmark (**Song et al.** 2015). This dataset consists of 10335 synchronized RGB-D pairs, where pixel-wise annotation is available. There is a standard trainval-test split, with 5050 images for testing and 5285 images for training/validation. The SUN RGB-D benchmark is a collection of images captured with different types of RGB-D cameras. The dataset also contains in-painted depth images, obtained by making use of multi-view fusion technique. In the experiments we also used the standard training and test split with in-painted depth images. However, we excluded 587 training images that are originally obtained with RealSense RGB-D camera. This is due to the fact that raw depth images from the aforementioned camera consist of many invalid measurements, therefore in-painted depth images have many false values. We remark that the SUN RGB-D dataset is highly unbalanced in terms of class instances, where 16 out of 37 classes rarely present. To prevent the network from over-fitting towards unbalanced class distribution, we weighted the loss for each class with the median frequency class balancing (**Eigen and Fergus** 2015). In particular, the class *floormat* and *shower-curtain* have the least frequencies and they are the most challenging ones in the segmentation task. Moreover, approximately 0.25% pixels are not annotated and do not belong to any of the target 37 classes.

6.4.0.1 Training

We trained the all networks end-to-end. Therefore images were resized to the resolution of 224×224. To this end we applied bilinear interpolation on the RGB images and nearest-neighbor interpolation on the depth images and the ground-truth labeling. The networks were implemented with the Caffe framework (**Jia et al.** 2014) and were trained with stochastic gradient descent (SGD) solver (**Bottou** 2012) using a batch size of 4. The input data was randomly shuffled after each epoch. The learning rate was initialized to 0.001 and was multiplied by 0.9 in every 50,000 iterations. We used a momentum of 0.9 and set weight decay to 0.0005. We trained the networks until convergence, when no further decrease in the loss was observed. The parameters in the encoder part of the network were fine-tuned from the VGG 16-layer model (**Simonyan and Zisserman** 2015) pre-trained on the ImageNet dataset (**Russakovsky et al.** 2015). The original VGGNet requires a three-channel color image. Therefore, for different input dimensions we processed the weights of first layer (*i.e.*, conv1_1) as follows:

1. averaged the weights along the channel for a single-channel depth input;
2. stacked the weights with their average for a four-channel RGB-D input;
3. duplicated the weights for a six-channel RGB-HHA input.

6.4.0.2 Testing

We evaluated the results on the original 5050 test images. For quantitative evaluation, we used the following three criteria. Let TP, FP, FN denote the total number of true positive, false positive, false negative and N denote the total valid (annotated) pixels, we define the following criteria:

1. Global accuracy, referred to as *global*, is the percentage of the correctly classified pixels, defined as
$$\text{global accuracy} := \frac{1}{N}\sum_{c=1}^{K} \mathrm{TP}_c \,.$$

2. Mean accuracy, referred to as *mean*, is the average of classwise accuracy, defined as
$$\text{classwise accuract} := \frac{1}{K}\sum_{c=1}^{K} \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c} \,.$$

3. Intersection-over-union (IoU), is average value of the intersection of the prediction and ground truth regions over the union of them, defined as
$$\text{classwise IoU} := \frac{1}{K}\sum_{c=1}^{K} \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c + \mathrm{FN}_c} \,.$$

Among these three measures, the global accuracy is relatively less informative due to the unbalanced class distribution. In general, the frequent classes receive a high score and hence dominate the less frequent ones. Therefore we also measured the average class accuracy and IoU score to provide a better evaluation of our method.

6.4.1 Quantitative Results

In the first experiment, we compared our FuseNet to the state-of-the-art methods. The results are presented in Table 6.1. We denote the SparseFusion and DenseFusion by FuseNet-SF, FuseNet-DF, respectively, following by the number of fusion layers used in the network (*e.g.*, , FuseNet-SF5). The results shows that our FuseNet outperforms most of the existing methods with a significant margin. FuseNet is not as competitive in comparison to the Context-CRF by **G. Lin et al.** (2016a). However, it is also worth noting that the Context-CRF paper trains the network with a different loss function that corresponds to piecewise CRF training. It also requires mean-field approximation at the inference stage, followed by a dense fully connected CRF refinement to produce the final prediction. Applying the similar loss function and post-processing, FuseNet is likely to produce on-par or better results.

In the second experiment, we compare the FuseNet to network trained with different represenation of depth, in order to futher evaluate the effectiveness of depth fusion and different fusion variations. The results are presented in Table 6.2. The upper part of the table provides the results of network trained with RGB, depth and different combination and representation of depth. It can be seen that stacking depth and HHA into color gives slight improvements over network trained with only color, depth or HHA. In contrast, with the depth fusion of FuseNet, we improve over a significant margin, in particular with respect to the IoU scores. We remark that the depth fusion is in particular useful as a replacement for HHA. Instead of preprocessing a single channel depth images to obtain hand crafted

Table 6.1: Segmentation results on the SUN RGB-D dataset (37 classes) in comparison to the state of the art. Our methods FuseNet-DF1 and FuseNet-SF5 outperforms most of the existing methods, except for the Context-CRF by **G. Lin et al.** (2016a).

	global accuracy	classwise accuracy	classwise IoU
FCN-32s [123]	68.35	41.13	29.00
FCN-16s [123]	67.51	38.65	27.15
Bayesian SegNet (RGB) [97]	71.20	45.90	30.70
LSTM [114]	-	48.10	-
Context-CRF (RGB) [116]	78.40	53.40	42.30
FuseNet-SF5	76.27	48.30	37.29
FuseNet-DF1	73.37	50.07	34.02

Table 6.2: Segmentation results of FuseNet in comparison to the networks trained with RGB, depth, HHA and their combinations. The second part of the table provides the results of different variations of FuseNet. We show that FuseNet obtained significant improvments by extracting more informative features from depth.

	global accuracy	classwise accuracy	classwise IoU
Depth	69.06	42.80	28.49
HHA	69.21	43.23	28.88
RGB	72.14	47.14	32.47
RGB-D	71.39	49.00	31.95
RGB-HHA	73.90	45.57	33.64
FusetNet-SF1	75.48	46.15	35.99
FusetNet-SF2	75.82	46.44	36.11
FusetNet-SF3	76.18	47.10	36.63
FusetNet-SF4	76.56	48.46	37.76
FusetNet-SF5	76.27	48.30	37.29
FusetNet-DF1	73.37	50.07	34.02
FusetNet-DF2	73.31	49.39	33.97
FusetNet-DF3	73.37	49.46	33.52
FusetNet-DF4	72.83	49.53	33.46
FusetNet-DF5	72.56	49.86	33.04

three-channel HHA representation, FuseNet learns high dimensional features from depth end-to-end, which is more informative as shown by experiments.

With Table 6.2, we also analyzed the performance of different variations of FuseNet (shown in the lower part). Since the original VGG 16-layer network has 5 levels of pooling, we increase the number of fusion layers as the network gets deeper. The experiments show that segmentation accuracy gets improved from FuseNet-SF1 to FuseNet-SF5, however the increase appears saturated up to the fusion after the 4th pooling, *i.e.,* , FuseNet-SF4. The possible reason behind the accuracy saturation is that depth already provides very distinguished features at low-level to compensate textureless regions in RGB, and we consistently fuse features extracted from depth into the RGB-branch. The same trend can be observed with FuseNet-DF.

In the third experiment, we further compare FuseNet-SF5, FuseNet-DF1 to the network trained with RGB-D input. In Table 6.3 and Table 6.4, we report the classwise accuracy and IoU scores of 37 classes, respectively. For class accuracy, all the three network architectures give very comparable results. However, for IoU scores, FuseNet-SF5 outperforms in 30 out of 37 classes in comparison to other two networks. Since the classwise IoU is a better measurement over global and mean accuracy, FuseNet obtains significant improvements over the network trained with stacked RGB-D, showing that depth fusion is a better approach to extract informative features from depth and to combine them with color features. In Figure 6.5, we show some visual comparison of the FuseNet.

6.5 Conclusions

In this paper, we have presented a fusion-based CNN network for semantic labeling on RGB-D data. More precisely, we have proposed a solution to fuse depth information with RGB data by making use of a CNN. The proposed network has an encoder-decoder type architecture, where the encoder part is composed of two branches of networks that simultaneously extract features from RGB and depth channels. These features are then fused into the RGB feature maps as the network goes deeper.

By conducting a comprehensive evaluation, we may conclude that the our approach is a competitive solution for semantic segmentation on RGB-D data. The proposed FuseNet outperforms the current CNN-based networks on the challenging SUN RGB-D benchmark (**Song et al.** 2015). We have also investigated two possible fusion approaches, *i.e.,* dense fusion and sparse fusion. By applying the latter one with a single fusion operation we have obtained a slightly better performance. Nevertheless we may conclude that both fusion approaches provide similar results. Interestingly, we can also claim that HHA representation itself provides a superficial improvement to the depth information. We also remark that a straight-forward extension of the proposed approach can be applied for other classification tasks such as image or scene classification.

Table 6.3: Classwise segmentation accuracy of 37 classes. We compare FuseNet-SF5, FuseNet-DF1 to the network trained with stacked RGB-D input.

	🟥 wall	🟩 floor	🟨 cabin	🟦 bed	🟪 chair	🟦 sofa	⬜ table	🟫 door	🟥 wdw	🟩 bslf
RGB-D	77.19	93.90	**62.51**	74.62	71.22	59.09	**66.76**	42.27	62.73	29.51
FuseNet-SF5	**90.20**	**94.91**	61.81	**77.10**	**78.62**	**66.49**	65.44	**46.51**	62.44	**34.94**
FuseNet-DF1	82.39	93.88	56.97	73.76	78.02	62.85	60.60	45.43	**67.22**	28.79
	🟦 desk	🟥 shelf	🟩 ctn	🟫 drssr	🟩 pillow	🟩 mirror	🟦 mat	🟪 clthes	🟩 ceil	🟩 books
RGB-D	12.12	9.27	63.26	40.44	52.02	52.99	0.00	**38.38**	84.06	**57.05**
FuseNet-SF5	**25.63**	**20.28**	**65.94**	44.03	54.28	52.47	0.00	25.89	84.77	45.23
FuseNet-DF1	20.98	14.46	61.43	**48.63**	**58.59**	**55.96**	0.00	30.52	**86.23**	53.86
	🟨 towel	🟪 shwr	🟥 box	🟩 board	🟨 person	⬛ stand	🟥 toilet	🟩 sink	🟨 lamp	🟦 btub
RGB-D	**27.92**	4.99	**31.24**	69.08	16.97	42.70	76.80	**69.41**	50.28	65.41
FuseNet-SF5	21.05	**8.82**	21.94	57.45	19.06	37.15	76.77	68.11	49.31	73.23
FuseNet-DF1	27.14	1.96	26.61	**66.36**	**30.91**	**43.89**	**81.38**	66.47	**52.64**	**74.73**
	🟧 pic	🟪 cnter	🟥 blinds	🟫 fridge	🟧 tv	🟩 paper	🟪 bag	mean		
RGB-D	64.66	**48.19**	**48.80**	**34.90**	45.77	**41.54**	24.90	49.00		
FuseNet-SF5	67.39	40.37	43.48	34.52	34.83	24.08	12.62	48.30		
FuseNet-DF1	**67.50**	39.89	44.73	32.31	**53.13**	36.67	**25.80**	**50.07**		

Table 6.4: Classwise IoU scores of 37 classes. We compare FuseNet-SF5, FuseNet-DF1 to the network trained with stacked RGB-D input.

	■ wall	■ floor	■ cabin	■ bed	■ chair	■ sofa	■ table	■ door	■ wdw	■ bslf
RGB-D	69.46	86.10	35.56	58.29	60.02	43.09	46.37	27.76	43.3	19.70
FuseNet-SF5	**74.94**	**87.41**	**41.70**	**66.53**	**64.45**	**50.36**	**49.01**	**33.35**	**44.77**	**28.12**
FuseNet-DF1	69.48	86.09	35.57	58.27	60.03	43.09	46.38	27.78	43.31	19.75

	■ desk	■ shelf	■ ctn	■ drssr	■ pillow	■ mirror	■ mat	■ clthes	■ ceil	■ books
RGB-D	10.19	5.34	43.02	23.93	30.70	31.00	0.00	**17.67**	63.10	21.79
FuseNet-SF5	**18.31**	**9.20**	**52.68**	**34.61**	**37.77**	**38.87**	0.00	16.67	**67.34**	**27.29**
FuseNet-DF1	15.61	7.44	42.24	28.74	31.99	34.73	0.00	15.82	60.09	24.28

	■ towel	■ shwr	■ box	■ board	■ person	■ stand	■ toilet	■ sink	■ lamp	■ btub
RGB-D	13.21	4.13	14.21	40.43	10.00	11.79	59.17	45.85	26.06	51.75
FuseNet-SF5	**16.55**	**6.06**	**15.77**	49.23	14.59	**19.55**	**67.06**	**54.99**	**35.07**	**63.06**
FuseNet-DF1	13.6	1.54	15.47	**45.21**	**15.49**	17.46	63.38	48.09	27.06	56.85

	■ pic	■ cnter	■ blinds	■ fridge	■ tv	■ paper	■ bag	mean
RGB-D	36.24	25.48	29.11	22.69	31.31	12.05	12.38	31.95
FuseNet-SF5	**46.84**	**27.73**	**31.47**	**31.31**	31.64	16.01	9.52	**37.29**
FuseNet-DF1	36.30	25.44	29.12	23.63	**37.67**	**16.45**	**12.92**	34.02

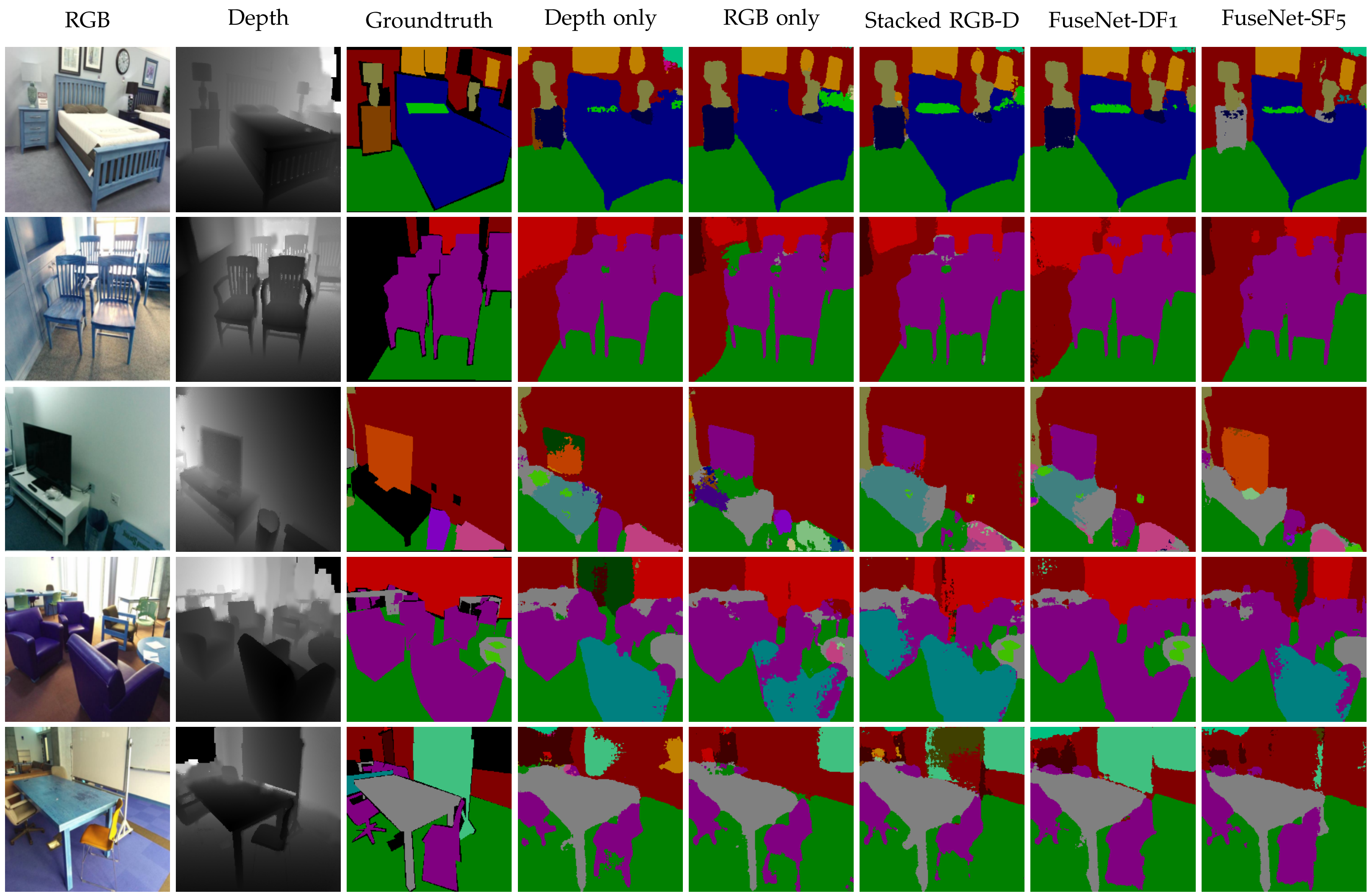


Figure 6.5: Qualitative segmentation results for different architectures. The first three columns contain RGB and depth images along with the groundtruth, respectively, followed by the segmentation results. Last two columns contain the results obtained by our FuseNet-DF1 and FuseNet-SF5 approaches. The color notation is as indicated in Table 6.3 and Table 6.4.

CHAPTER 7

Multi-View Deep Learning for Consistent Semantic Mapping with RGB-D Cameras

ABSTRACT Visual scene understanding is an important capability that enables robots to purposefully act in their environment. In this paper, we propose a novel deep neural network approach to predict semantic segmentation from RGB-D sequences. The key innovation is to train our network to predict multi-view consistent semantics in a self-supervised way. At test time, its semantics predictions can be fused more consistently in semantic keyframe maps than predictions of a network trained on individual views. We base our network architecture on a recent single-view deep learning approach to RGB and depth fusion for semantic object-class segmentation and enhance it with multi-scale loss minimization. We obtain the camera trajectory using RGB-D SLAM and warp the predictions of RGB-D images into ground-truth annotated frames in order to enforce multi-view consistency during training. At test time, predictions from multiple views are fused into keyframes. We propose and analyze several methods for enforcing multi-view consistency during training and testing. We evaluate the benefit of multi-view consistency training and demonstrate that pooling of deep features and fusion over multiple views outperforms single-view baselines on the NYUDv2 benchmark for semantic segmentation. Our end-to-end trained network achieves state-of-the-art performance on the NYUDv2 dataset in single-view segmentation as well as multi-view semantic fusion.

© 2017 IEEE. Reprinted, with permission, from Lingni Ma, Jörg Stückler, Christian Kerl and Daniel Cremers, Multi-view Deep Learning for Consistent Semantic Mapping with RGB-D Cameras, in proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sept. 2017. Revised layout, minor adaptations and extended visualization for Figure 7.4, Figure 7.5.

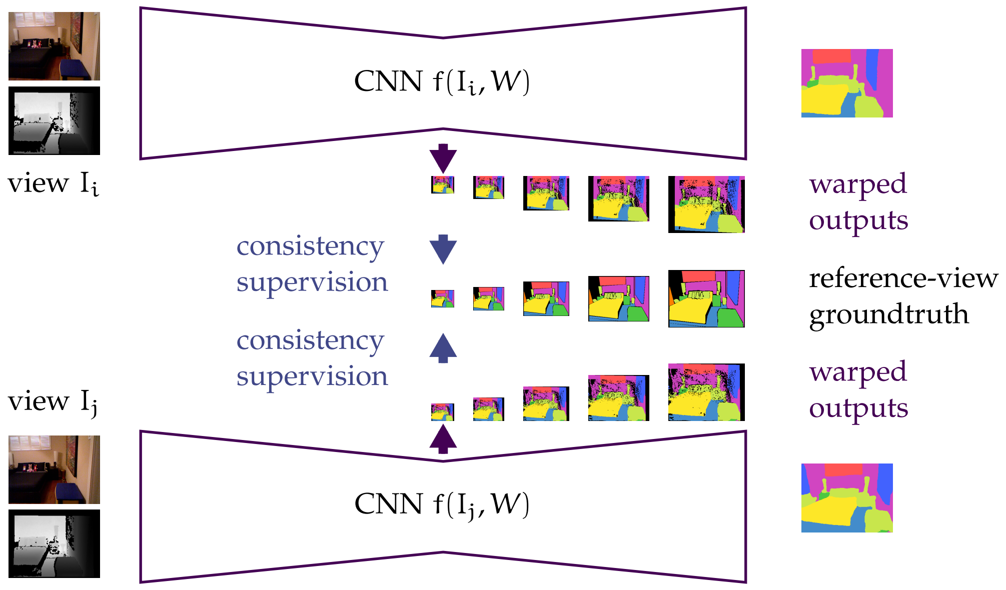


Figure 7.1: We train our CNN to predict multi-view consistent semantic segmentations for RGB-D images. The key innovation is to enforce consistency by warping CNN feature maps from multiple views into a common reference view using the SLAM trajectory and to supervise training at multiple scales. Our approach improves performance for single-view segmentation and is specifically beneficial for multi-view fused segmentation.

7.1 Introduction

Intelligent robots require the ability to understand their environment through parsing and segmenting the 3D scene into meaningful objects. The rich appearance-based information contained in images renders vision a primary sensory modality for this task.

In recent years, large progress has been achieved in semantic segmentation of images. Most current state-of-the-art approaches apply deep learning for this task. With RGB-D cameras, appearance as well as shape modalities can be combined to improve the semantic segmentation performance. Less explored, however, is the usage and fusion of multiple views onto the same scene which appears naturally in the domains of 3D reconstruction and robotics. Here, the camera is moving through the environment and captures the scene from multiple view points. Semantic SLAM aims at aggregating several views in a consistent 3D geometric and semantic reconstruction of the environment.

In this paper, we propose a novel deep learning approach for semantic segmentation of RGB-D images with multi-view context. We base our network on a recently proposed deep convolutional neural network (CNN) for RGB and depth fusion by **Hazirbas et al.** (2017) and enhance the approach with multi-scale deep supervision. Based on the trajectory obtained through RGB-D simultaneous localization and mapping (SLAM), we further regularize the CNN training with multi-view consistency constraints as shown in Figure 7.1. We propose and evaluate several variants to enforce multi-view consistency during training. A shared principle is using the SLAM trajectory estimate to warp network outputs of multiple frames into the reference view with ground-truth annotation. By this, the network not only learns features that are invariant under view-point change. Our semi-supervised training approach also makes better use of the annotated ground-truth data than single-view learning. This alleviates the need for large amounts of annotated training data which is expensive to obtain. Complementary to our training approach, we aggregate the predictions of our trained network in keyframes to increase segmentation accuracy at testing. The predictions of neighboring images are fused into the keyframe based on the SLAM estimate in a probabilistic way.

In experiments, we evaluate the performance gain achieved through multi-view training and fusion at testing over single-view approaches. Our results demonstrate that multi-view max-pooling of feature maps during training best supports multi-view fusion at testing. Overall we find that enforcing multi-view consistency during training significantly improves fusion at test time versus fusing predictions from networks trained on single views. Our end-to-end training achieves state-of-the-art performance on the NYUDv2 dataset in single-view segmentation as well as multi-view semantic fusion. While the fused keyframe segmentation can be directly used in robotic perception, our approach can also be useful as a building block for semantic SLAM using RGB-D cameras.

7.2 Related Work

Recently, remarkable progress has been achieved in semantic image segmentation using deep neural networks and, in particular, CNNs. On many benchmarks, these approaches excell previous techniques by a great margin.

Image-based Semantic Segmentation. As one early attempt, **Couprie et al.** (2013) propose a multiscale CNN architecture to combine information at different receptive field resolutions and achieved reasonable segmentation results. **Gupta et al.** (2014) integrate depth into the R-CNN approach by **Girshick et al.** (2014) to detect objects in RGB-D images. They convert depth into 3-channel HHA, *i.e.,* disparity, height and angle encoding and achieve semantic segmentation by training a classifier for superpixels based on the CNN features. **Long et al.** (2015) propose a fully convolutional network (FCN) which enables end-to-end training for semantic segmentation. Since CNNs reduce the input spatial resolution by a great factor through layers pooling, FCN presents an upsample stage to output high-resolution segmentation by fusing low-resolution predictions. Inspired by FCN and auto-encoders (**Bengio et al.** 2007), encoder-decoder architectures have been proposed to learn upsampling with unpooling and deconvolution (**Noh et al.** 2015). For RGB-D images, **Eigen and Fergus** (2015) propose to train CNNs to predict depth, surface normals and semantics with a multi-task network and achieve very good performance. FuseNet (**Hazirbas et al.** 2017) proposes an encoder-decoder CNN to fuse color and depth cues in an end-to-end training for semantic segmentation, which is shown to be more efficient in learning RGB-D

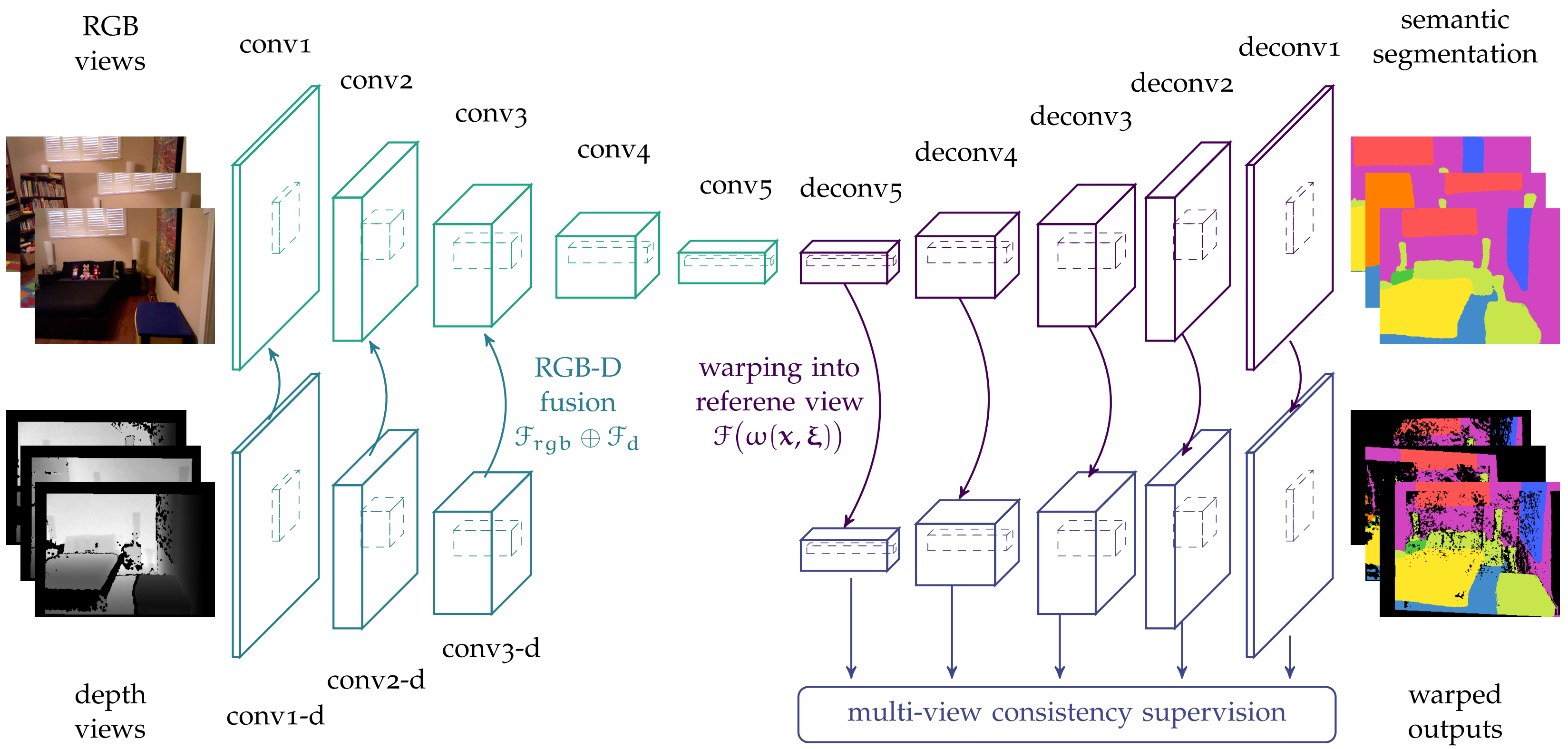


Figure 7.2: The CNN encoder-decoder architecture used in our approach. Input to the network are RGB-D sequences with corresponding poses from SLAM trajectory. The encoder contains two branches to learn features from RGB-D data as inspired by FuseNet **Hazirbas et al.** (2017). The obtained low-resolution high-dimension feature maps are successively refined through deconvolutions in the decoder. We warp feature maps into a common reference view and enforce multi-view consistency with various constraints. The network is trained in a deeply-supervised manner where loss is computed at all scales of the decoder.

features in comparison to direct concatenation of RGB and depth or the use of HHA. Recently, more complex CNN architectures have been proposed that include multi-resolution refinement by **G. Lin et al.** (2017), dilated convolutions by **Yu and Koltun** (2016) and residual units (*e.g.,* **Wu et al.** (2016)) to achieve state-of-the-art single image semantic segmentation. **Z. Li et al.** (2016a) use a LSTM recurrent neural network to fuse RGB and depth cues and obtain smooth predictions. **G. Lin et al.** (2016a); **G. Lin et al.** (2016b) design a CNN that corresponds to a conditional random field (CRF) and use piecewise training to learn both unary and pairwise potentials end-to-end. Our approach trains a network on multi-view consistency and fuses the results from multiple view points. It is complementary to the above single-view CNN approaches.

Semantic SLAM. In the domain of semantic SLAM, **Salas-Moreno et al.** (2013) developed the SLAM++ algorithm to perform RGB-D tracking and mapping at the object instance level. **Hermans et al.** (2014) proposed 3D semantic mapping for indoor RGB-D sequences based on RGB-D visual odometry and a random forest classifier that performs semantic image segmentation. The individual frame segmentations are projected into 3D and smoothed using a CRF on the point cloud. **Stückler et al.** (2015) perform RGB-D SLAM and probabilistically fuse the semantic segmentations of individual frames obtained with a random forest in multi-resolution voxel maps. Recently, **Armeni et al.** (2016) propose a hierarchical parsing method for large-scale 3D point clouds of indoor environments. They first seperate point clouds into disjoint spaces, *i.e.,* single rooms, and then further cluster points at the object level according to handcrafted features.

Multi-View Semantic Segmentation. In contrast to the popularity of CNNs for image-based segmentation, it is less common to apply CNNs for semantic segmentation on multi-view 3D reconstructions. Recently, **Riegler et al.** (2016) apply 3D CNNs on sparse octree data structures to perform semantic segmentation on voxels. Nevertheless, the volumetric representations may discard details which are present at the original image resolution. **McCormac et al.** (2016) proposed to fuse CNN semantic image segmentations on a 3D surfel map (**Whelan et al.** 2016). **Y. He et al.** (2017) propose to fuse CNN semantic segmentations from multiple views in video using superpixels and optical flow information. In contrast to our approach, these methods do not impose multi-view consistency during CNN training and cannot leverage the view-point invariant features learned by our network. **Kundu et al.** (2016) extend dense CRFs to videos by associating pixels temporally using optical flow and optimizing their feature similarity. Closely related to our approach for enforcing multi-view consistency is the approach by **Su et al.** (2015) who investigate the task of 3D shape recognition. They render multiple views onto 3D shape models which are fed into a CNN feature extraction stage that is shared across views. The features are max-pooled across view-points and fed into a second CNN stage that is trained for shape recognition. Our approach uses multi-view pooling for the task of semantic segmentation and is trained using realistic imagery and SLAM pose estimates. Our trained network is able to classify single views, but we demonstrate that multi-view fusion using the network trained on multi-view consistency improves segmentation performance over single-view trained networks.

7.3 CNN Architecture for Semantic Segmentation

In this section, we detail the CNN architecture for semantic segmentation of each RGB-D image of a sequence. We base our encoder-decoder CNN on FuseNet by **Hazirbas et al.** (2017) which learns rich features from RGB-D data. We enhance the approach with multi-scale loss minimization, which gains additional improvement in segmentation performance.

7.3.1 RGB-D Semantic Encoder-Decoder

Figure 7.2 illustrates our CNN architecture. The network follows an encoder-decoder design, similar to previous work on semantic segmentation (**Noh et al.** 2015). The encoder extracts a hierarchy of features through convolutional layers and aggregates spatial information by pooling layers to increase the receptive field. The encoder outputs low-resolution high-dimensional feature maps, which are upsampled back to the input resolution by the decoder through layers of memorized unpooling and deconvolution. Following FuseNet (**Hazirbas et al.** 2017), the network contains two branches to learn features from RGB ($\mathcal{F}_{rgb}$) and depth ($\mathcal{F}_{d}$), respectively. The feature maps from the depth branch are consistently fused into the RGB branch at each scale. We denote the fusion by $\mathcal{F}_{rgb} \oplus \mathcal{F}_{d}$.

The semantic label set is denoted as $L = \{1, 2, \ldots, K\}$ and the category index is indicated with subscript j. Following notation convention, we compute the classification score $S = (s_1, s_2, \ldots, s_K)^T$ at location x and map it to the probability distribution $P = (p_1, p_2, \ldots, p_K)^T$ with the softmax function $\sigma(\cdot)$. Network inference obtains the probability

$$p_j(x, W \mid I) = \sigma\left(s_j(x, W)\right) = \frac{\exp\left(s_j(x, W)\right)}{\sum_{k=1}^{K} \exp\left(s_k(x, W)\right)}, \tag{7.1}$$

of all pixels x in the image for being labeled as class j, given input RGB-D image I and network parameters W. We use the cross-entropy loss to learn network parameters for semantic segmentation from ground-truth annotations l_{gt},

$$L(W) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} [\![j = l_{gt}]\!] \log p_j\left(x_i, W \mid I\right), \tag{7.2}$$

where N is the number of pixels. This loss minimizes the Kullback-Leibler (KL) divergence between predicted distribution and the ground-truth, assuming the ground-truth has a one-hot distribution on the true label.

7.3.2 Multi-Scale Deep Supervision

The encoder of our network contains five 2×2 pooling layers and downsamples the input resolution by a factor of 32. The decoder learns to refine the low resolution back to the

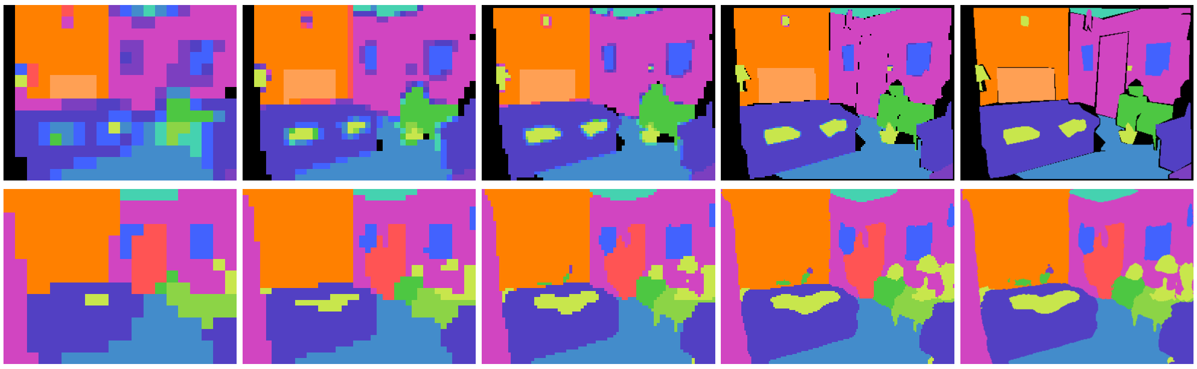


Figure 7.3: Example of multi-scale ground-truth and predictions. Upper row: successive subsampled of ground-truth annotation obtained through stochastic pooling. Lower row: CNN prediction on each scale. The resolutions are coarse to fine from left to right with 20×15, 40×30, 80×60, 160×120 and 320×240.

original one with five memorized unpooling followed by deconvolution. In order to guide the decoder through the successive refinement, we adopt the deeply supervised learning method (**Dosovitskiy et al.** 2015a; **Lee et al.** 2015) and compute the loss for all upsample scales. For this purpose, we append a classification layer at each deconvolution scale and compute the loss for the respective resolution of ground-truth which is obtained through stochastic pooling (**M. Zeiler and Fergus** 2013) over the full resolution annotation (see Figure 7.3 for an example).

7.4 Multi-View Consistent Learning and Prediction

While CNNs have been shown to obtain the state-of-the-art semantic segmentation performances for many datasets, most of these studies focus on single views. When observing a scene from a moving camera such as on a mobile robot, the system obtains multiple different views onto the same objects. The key innovation of this work is to explore the use of temporal multi-view consistency within RGB-D sequences for CNN training and prediction. For this purpose, we perform 3D data association by warping multiple frames into a common reference view. This then enables us to impose multi-view constraints during training. In this section, we describe several variants of such constraints. Notably, these methods can also be used at test time to fuse predictions from multiple views in a reference view.

7.4.1 Multi-view Data Association Through Warping

Instead of single-view training, we train our network on RGB-D sequences with poses estimated by a SLAM algorithm. We define each training sequence to contain one reference view I_k with ground-truth semantic annotations and several overlapping views I_i that are tracked towards I_k. The relative poses ξ of the neighboring frames are estimated through tracking algorithms such as DVO SLAM (**Kerl et al.** 2013a). In order to impose temporal consistency, we adopt the warping concept from multi-view geometry to associate pixels between view points and introduce warping layers into our CNN. The warping layers synthesize CNN output in a reference view from a different view at any resolution by sampling given a known pose estimate and the known depth. The warping layers can be viewed as a variant of spatial transformers by **Jaderberg et al.** (2015) with fixed transformation parameters.

We now formulate the warping. Given 2D image coordinate $x \in \mathbb{R}^2$, the warped pixel location

$$x^{\omega} = \omega(x, \xi) = \pi\left(\mathbf{T}(\xi)\,\pi^{-1}\left(x, Z_i(x)\right)\right) \ , \tag{7.3}$$

is determined through the warping function $\omega(x, \xi)$ which transforms the location from one camera view to the other using the depth $Z_i(x)$ at pixel x in image I_i and the SLAM pose estimate ξ. The functions π and its inverse π^{-1} project homogeneous 3D coordinates to image coordinates and vice versa, while $\mathbf{T}(\xi)$ denotes the homogeneous transformation matrix derived from pose ξ.

Using this association by warping, we synthesize the output of the reference view by sampling the feature maps of neighboring views using bilinear interpolation. Since the interpolation is differentiable, it is straight-forward to back-propagate gradients through the warping layers. With a slight abuse of notation, we denote the operation of synthesizing the layer output $\mathcal{F}$ given the warping by $\mathcal{F}^{\omega} := \mathcal{F}\left(\omega(x, \xi)\right)$.

We also apply deep supervision when training for multi-view consistency through warping. As shown in Figure 7.2, feature maps at each resolution of the decoder are warped into the common reference view. Despite the need to perform warping at multiple scales, the warping grid is only required to be computed once at the input resolution, and is normalized to the canonical coordinates within the range of $[-1, 1]$. The lower-resolution warping grids can then be efficiently generated through average pooling layers.

7.4.2 Consistency Through Warp Augmentation

One straight-forward solution to enforce multi-view segmentation consistency is to warp the predictions of neighboring frames into the ground-truth annotated keyframe and computing a supervised loss there. This approach can be interpreted as a type of data augmentation using the available nearby frames. We implement this consistency method by warping the keyframe into neighboring frames, and synthesize the classification score of the nearby frame from the keyframe's view point. We then compute the cross-entropy loss on this synthesized prediction. Within RGB-D sequences, objects can appear at various scales, image locations, view perspective, color distortion given uncontrolled lighting and shape distortion given rolling shutters of RGB-D cameras. Propagating the keyframe annotation into other frames implicitly regulates the network predictions to be invariant under these transformations.

7.4.3 Consistency Through Bayesian Fusion

Given a sequence of measurements and predictions at test time, Bayesian fusion is frequently applied to aggregate the semantic segmentations of individual views. Let us denote the semantic labeling of a pixel by y and its measurement in frame i by z_i. We use the notation z^i for the set of measurements up to frame i. According to Bayes rule,

$$\begin{aligned} p(y \mid z^i) &= \frac{p(z_i \mid y, z^{i-1})\, p(y \mid z^{i-1})}{p(z_i \mid z^{i-1})} && (7.4) \\ &= \eta_i\, p(z_i \mid y, z^{i-1})\, p(y \mid z^{i-1}) \,. && (7.5) \end{aligned}$$

Suppose measurements satisfy the *i.i.d.* condition, *i.e.*, $p(z_i \mid y, z^{i-1}) = p(z_i \mid y)$, and equal a-priori probability for each class, then Equation (7.4) simplifies to

$$p(y \mid z^i) = \eta_i\, p(z_i \mid y)\, p(y \mid z^{i-1}) = \prod_i \eta_i\, p(z_i \mid y) \,. \tag{7.6}$$

Put simple, Bayesian fusion can be implemented by taking the product over the semantic labeling likelihoods of individual frame at a pixel and normalizing the product to yield a valid probability distribution. This process can also be implemented recursively on a sequence of frames.

When training our CNN for multi-view consistency using Bayesian fusion, we warp the predictions of neighboring frames into the keyframe using the SLAM pose estimate. We obtain the fused prediction at each keyframe pixel by summing up the unnormalized log labeling likelihoods instead of the individual frame softmax outputs. Applying softmax on the sum of log labeling likelihoods yields the fused labeling distribution. This is equivalent to Equation (7.6) since

$$\frac{\prod_i p_{ij}^{\omega}}{\sum_k^K \prod_i p_{ik}^{\omega}} = \frac{\prod_i \sigma\left(s_{ij}^{\omega}\right)}{\sum_k^K \prod_i \sigma\left(s_{ik}^{\omega}\right)} = \sigma\left(\sum_i s_{ij}^{\omega}\right) \,, \tag{7.7}$$

where s_{ij}^{ω} and p_{ij}^{ω} denote the warped classification scores and probabilities, respectively, and $\sigma(\cdot)$ is the softmax function as defined in Equation (7.1).

7.4.4 Consistency Through Multi-View Max-Pooling

While Bayesian fusion provides an approach to integrate several measurements in the probability space, we also explore direct fusion in the feature space using multi-view max-pooling of the warped feature maps. We warp the feature maps preceeding the classification layers at each scale in our decoder into the keyframe and apply max-pooling over corresponding feature activations at the same warped location to obtain a pooled feature map in the keyframe,

$$\mathcal{F} = \text{max\_pool}\left(\mathcal{F}_1^{\omega}, \mathcal{F}_2^{\omega}, \ldots, \mathcal{F}_N^{\omega}\right) \,. \tag{7.8}$$

The fused feature maps are classified and the resulting semantic segmentation is compared to the keyframe ground-truth for loss calculation.

Table 7.1: Single-view semantic segmentation accuracy of our network in comparison to the state-of-the-art methods for NYUDv2 13-class and 40-class segmentation tasks.

	methods	input	pixelwise	classwise	IoU
NYUDv2 13 classes	Couprie et al. [37]	RGB-D	52.4	36.2	-
	Hermans et al. [81]	RGB-D	54.2	48.0	-
	SceneNet [72]	DHA	67.2	52.5	-
	Eigen et al. [49]	RGB-D-N	75.4	66.9	52.6
	FuseNet-SF3 [3]	RGB-D	75.8	66.2	54.2
	MVCNet-Mono	RGB-D	77.6	68.7	56.9
	MVCNet-Augment	RGB-D	77.6	69.3	57.2
	MVCNet-Bayesian	RGB-D	**77.8**	*69.4*	**57.3**
	MVCNet-MaxPool	RGB-D	*77.7*	**69.5**	**57.3**
NYUDv2 40 classes	RCNN [69]	RGB-HHA	60.3	35.1	28.6
	FCN-16s [123]	RGB-HHA	65.4	46.1	34.0
	Eigen et al. [49]	RGB-D-N	65.6	45.1	34.1
	FuseNet-SF3 [3]	RGB-D	66.4	44.2	34.0
	Context-CRF [116]	RGB	67.6	49.6	37.1
	MVCNet-Mono	RGB-D	*68.6*	48.7	37.6
	MVCNet-Augment	RGB-D	*68.6*	*49.9*	**38.0**
	MVCNet-Bayesian	RGB-D	68.4	49.5	37.4
	MVCNet-MaxPool	RGB-D	**69.1**	**50.1**	**38.0**

7.5 Evaluation

We evaluate our proposed approach using the NYUDv2 RGB-D dataset (**Silberman et al.** 2012). The dataset provides 1449 pixelwise annotated RGB-D images capturing various indoor scenes, and is split into 795 frames for training/validation (trainval) and 654 frames for testing. The original sequences that contain these 1449 images are also available with NYUDv2, whereas sequences are unfortunately not available for other large RGB-D semantic segmentation datasets. Using DVO-SLAM (**Kerl et al.** 2013a), we determine the camera poses of neighboring frames around each annotated keyframe to obtain multi-view sequences. This provides us with in total 267,675 RGB-D images, despite that tracking fails for 30 out of 1449 keyframes. Following the original trainval/test split, we use 770 sequences with 143,670 frames for training and 649 sequences with 124,005 frames for testing. For benchmarking, our method is evaluated for the 13-class (**Couprie et al.** 2013) and 40-class (**Gupta et al.** 2013) semantic segmentation tasks. We use the raw depth images without inpainted missing values.

Table 7.2: Multi-view segmentation accuracy of our network using Bayesian fusion for NYUDv2 13-class and 40-class segmentation.

	methods	pixelwise	classwise	IoU
NYUDv2 13 classes	FuseNet-SF3 [3]	77.19	67.46	56.01
	MVCNet-Mono	78.70	69.61	58.29
	MVCNet-Augment	78.94	*70.48*	58.93
	MVCNet-Bayesian	**79.13**	*70.48*	*59.04*
	MVCNet-MaxPool	**79.13**	**70.59**	**59.07**
NYUDv2 40 classes	FuseNet-SF3 [3]	67.74	44.92	35.36
	MVCNet-Mono	70.03	49.73	39.12
	MVCNet-Augment	*70.34*	*51.73*	**40.19**
	MVCNet-Bayesian	70.24	51.18	39.74
	MVCNet-MaxPool	**70.66**	**51.78**	*40.07*

7.5.1 Training Details

We implemented our approach using the Caffe framework (**Jia et al.** 2014). For all experiments, the network parameters are initialized as follows. The convolutional kernels in the encoder are initialized with the pre-trained 16-layer VGGNet (**Simonyan and Zisserman** 2015) and the deconvolutional kernels in the decoder are initialized using He's method (**K. He et al.** 2015). For the first layer of the depth encoder, we average the original three-channel VGG weights to obtain a single-channel kernel. We train the network with stochastic gradient descent (SGD) (**Bottou** 2012) with 0.9 momentum and 0.0005 weight decay. The learning rate is set to 0.001 and decays by a factor of 0.9 every 30k iterations.

All the images are resized to a resolution of 320×240 pixels as input to the network and the predictions are also up to this scale. To downsample, we use cubic interpolation for RGB images and nearest-neighbor interpolation for depth and label images. During training, we use a mini-batch of 6 that comprises two sequences, with one keyframe and two tracking frames for each sequence. We apply random shuffling after each epoch for both inter and intra sequences. The network is trained until convergence. We observed that multi-view CNN training does not require significant extra iterations for convergence. For multi-view training, we sample from the nearest frames first and include 10 further-away frames every 5 epochs. By this, we alleviate that typically tracking errors accumulate and image overlap decreases as the camera moves away from the keyframe.

7.5.2 Evaluation Criteria

We measure the semantic segmentation performance with three criteria: global pixelwise accuracy, average classwise accuracy and average intersection-over-union (IoU) scores. These three criteria can be calculated from the confusion matrix. With K classes, each entry of the $K \times K$ confusion matrix c_{ij} is the total amount of pixels belonging to class i that are predicted to be class j. The global pixelwise accuracy is computed by $\sum_i^K c_{ii} / \left(\sum_i^K \sum_j^K c_{ij} \right)$, the average classwise accuracy is computed by $\frac{1}{K} \sum_i^K \left(c_{ii} / \sum_j^K c_{ij} \right)$, and the average IoU score is calculated by $\frac{1}{K} \sum_i^K \left(c_{ii} / \left(\sum_i^K c_{ij} + \sum_j^K c_{ij} - c_{ii} \right) \right)$.

7.5.3 Single Frame Segmentation

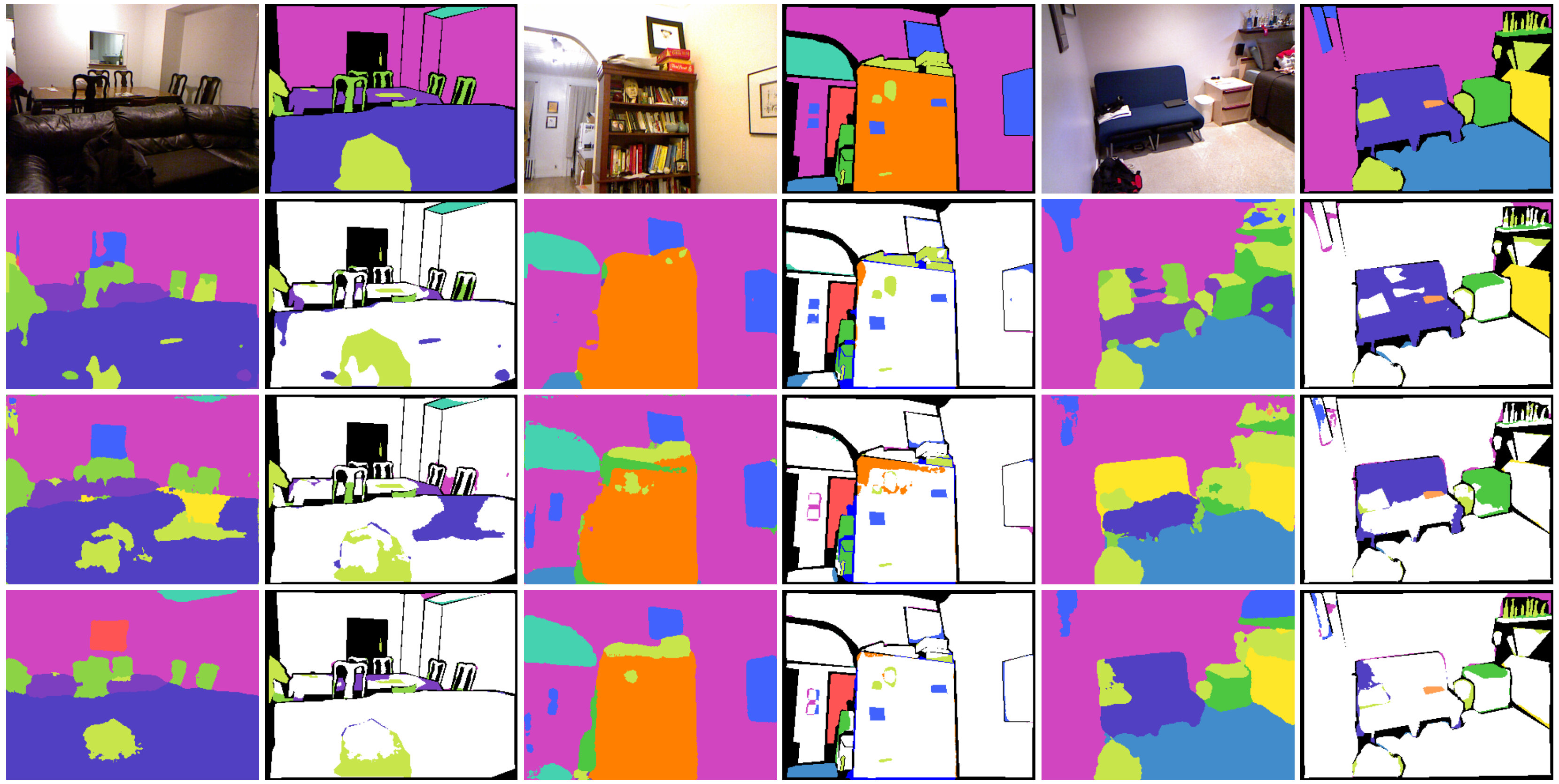
In a first set of experiments, we evaluate the performance of several variants of our network for direct semantic segmentation of individual frames. This means we do not fuse predictions from nearby frames to obtain the final prediction in a frame. We predict semantic segmentation with our trained models on the 654 test images of the NYUDv2 dataset and compare our methods with state-of-art approaches. The results are shown Table 7.1. Unless otherwise stated, we take the results from the original papers for comparison and report their best results (*i.e.,* SceneNet-FT-NYU-DO-DHA model for SceneNet (**Handa et al.** 2016), VGG-based model for **Eigen and Fergus** (2015)). The result of **Hermans et al.** (2014) is obtained after applying a dense CRF (**Krähenbühl and Koltun** 2011) for each image and in-between neighboring 3D points to further smoothen their results. We also remark that the results reported here for the Context-CRF model are finetuned on NYUDv2 like in our approach to facilitate comparison. Furthermore, the network output is refined using a dense CRF (**Krähenbühl and Koltun** 2011) which is claimed to increase the accuracy of the network by approximately 2%. The results for FuseNet-SF3 are obtained by our own implementation. Our baseline model MVCNet-Mono is trained without multi-view consistency, which amounts to FuseNet with multiscale deeply supervised loss at decoder. However, we apply single image augmentation to train the FuseNet-SF3 and MVCNet-Mono with random scaling between $[0.8, 1.2]$, random crop and mirror. This data augmentation is not used fro multi-view training. Nevertheless, our results show that the different variants of multi-view consistency training outperform the state-of-art methods for single image semantic segmentation. Overall, multi-view max-pooling (MVCNet-MaxPool) has a small advantage over the other multi-view consistency training approaches (MVCNet-Augment and MVCNet-Bayesian).

7.5.4 Multi-View Fused Segmentation

Since we train on sequences, in the second set of experiment, we also evaluate the fused semantic segmentation over the test sequences. The number of fused frames is fixed to 50, which are uniformly sampled over the entire sequence. Due to the lack of ground-truth for neighboring frames, we fuse the prediction of neighboring frames in the keyframes using Bayesian fusion according to Equation (7.7). This fusion is typically applied for semantic mapping using RGB-D SLAM. The results are shown Table 7.2. Bayesian multi-view fusion improves the semantic segmentation by approx. 2% on all evaluation measures towards single-view segmentation. Also, the training for multi-view consistency achieves a stronger

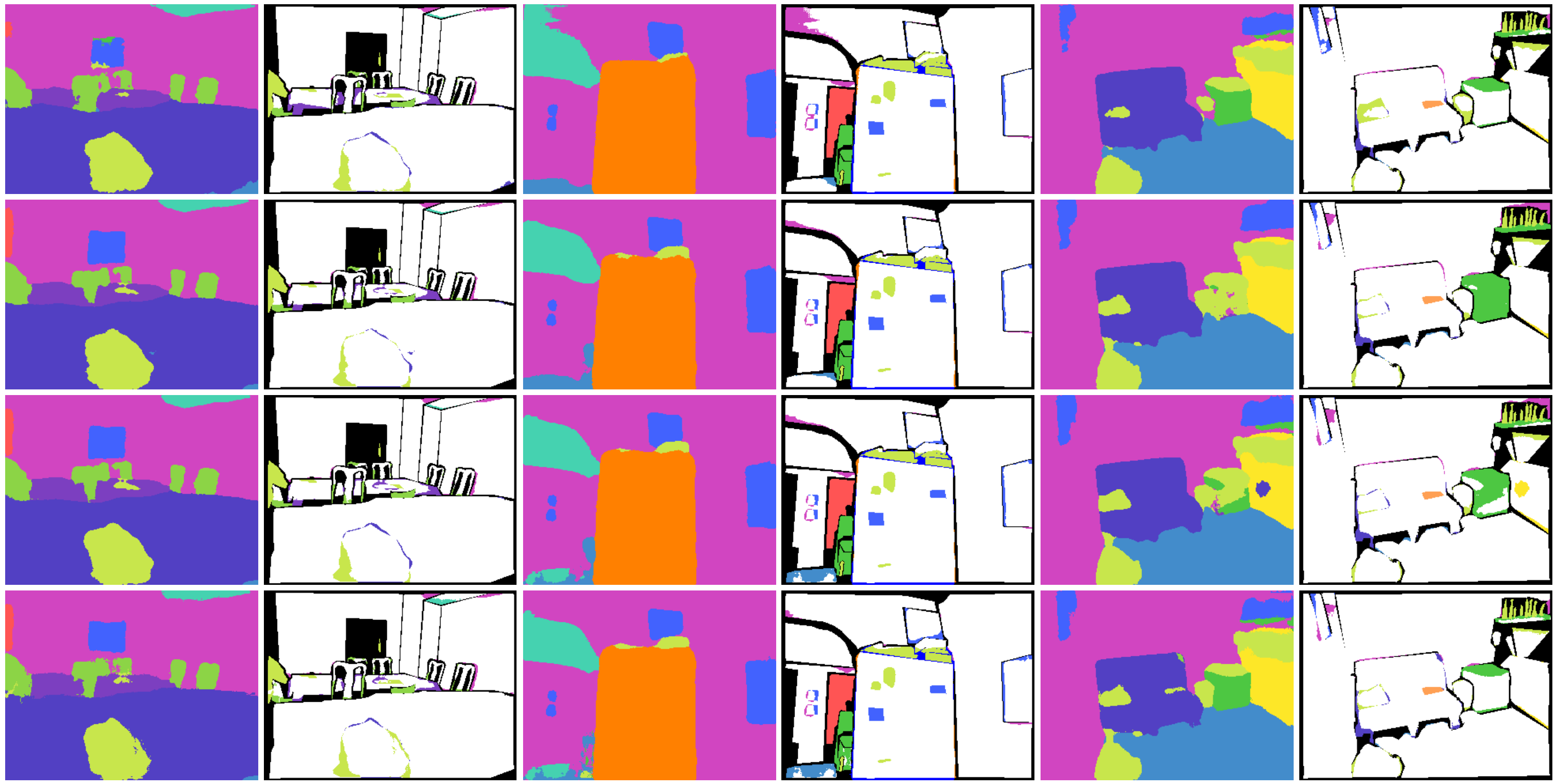
Table 7.3: NYUDv2 13-class semantic segmentation IoU scores. Our method achieves best per-class accuracy and average IoU.

	method	🟨 bed	🟩 objects	🟩 chair	🟩 furniture	🟩 ceiling	🟦 floor	🟦 decorat.	🟪 sofa	🟪 table	🟪 wall	🟥 window	🟧 books	🟧 TV	average accuracy
	class frequency	4.08	7.31	3.45	12.71	1.47	9.88	3.40	2.84	3.42	24.57	4.91	2.78	0.99	
single-view	Eigen et al. [49]	56.71	38.29	50.23	54.76	64.50	89.76	45.20	47.85	42.47	74.34	56.24	45.72	34.34	53.88
	FuseNet-SF3 [3]	61.52	37.95	52.67	53.97	64.73	89.01	47.11	57.17	39.20	75.08	58.06	37.64	29.77	54.14
	MVCNet-Mono	65.27	37.82	54.09	59.39	65.26	89.15	49.47	57.00	44.14	75.31	57.22	49.21	36.14	56.88
	MVCNet-Augment	65.33	38.30	54.15	**59.54**	**67.65**	89.26	49.27	55.18	43.39	74.59	58.46	**49.35**	**38.84**	57.18
	MVCNet-Bayesian	**65.76**	38.79	**54.60**	59.28	67.58	89.69	48.98	**56.72**	42.42	75.26	**59.55**	49.27	36.51	57.26
	MVCNet-MaxPool	65.71	**39.10**	54.59	59.23	66.41	**89.94**	**49.50**	56.30	**43.51**	**75.33**	59.11	49.18	37.37	**57.33**
multi-view	FuseNet-SF3 [3]	64.95	39.62	55.28	55.90	64.99	89.88	47.99	**60.17**	42.40	76.24	59.97	39.80	30.91	56.01
	MVCNet-Mono	67.11	40.14	56.39	60.90	66.07	89.77	50.32	59.49	46.12	76.51	59.03	48.80	37.13	58.29
	MVCNet-Augment	68.22	40.04	56.55	61.82	67.88	90.06	**50.85**	58.00	**45.98**	75.85	60.43	50.50	**39.89**	58.93
	MVCNet-Bayesian	**68.38**	**40.87**	**57.10**	**61.84**	**67.98**	**90.64**	50.05	59.70	44.73	76.50	**61.75**	**51.01**	36.99	59.04
	MVCNet-MaxPool	68.09	41.58	56.88	61.56	67.21	**90.64**	50.69	59.73	45.46	**76.68**	61.28	50.60	37.51	**59.07**



Rows from top to bottom: (a) input RGB image and groundtruth labelling, (b) predictions of **Eigen and Fergus** (2015), (c) predictions of FuseNet-SF3 **Hazirbas et al.** (2017), (d) predictions of our method MVCNet-Mono (baseline method without multi-view consistency constrains).

Figure 7.4: Qualitative semantic segmentation results of our methods and several state-of-the-art baselines on NYUDv2 13-class segmentation (see Table 7.3 for color coding, left columns: semantic segmentation, right columns: falsely classified pixels, black is void). Our multi-view consistency trained models produce more accurate and homogeneous results than single-view methods. Bayesian fusion further improves segmentation quality (*e.g.*, MVCNet-MaxPool-F). Continued on the following page.


(continued from the previous page) Rows from top to bottom: (a) predictions of our method MVCNet-Augment, (b) predictions of our method MVCNet-Bayesian, (c) predictions of our method MVCNet-MaxPool and (d) predictions of our method MVCNet-MaxPool-F.

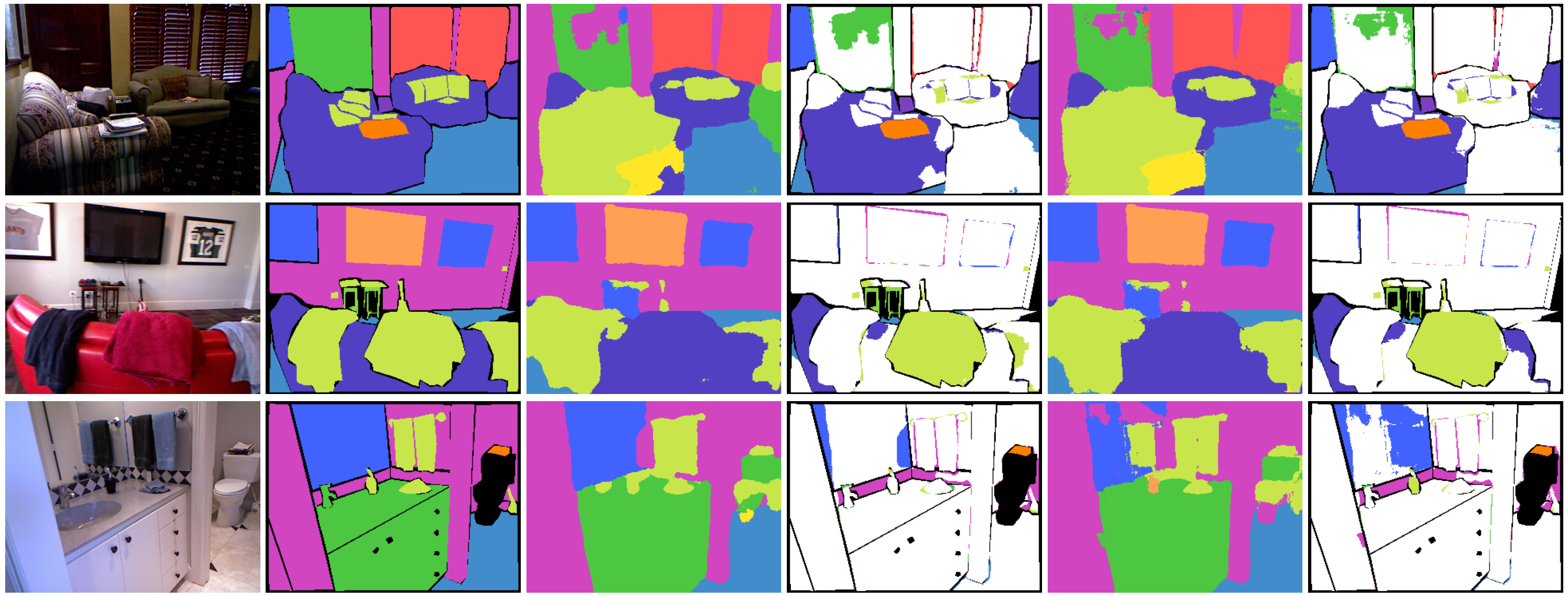


Figure 7.5: Challenging cases for MVCNet-MaxPool-F (from left column to right column: RGB image, ground-truth, single-view prediction on keyframe, multi-view prediction fused in keyframe). On the left, the network fails to classify the objects for all frames. In the middle, the network makes some errors in single-view prediction, but through multi-view fusion, some mistakes are corrected. On the right, multi-view fusion degenerates performance due to the mirror reflections.

gain over single-view training (MVCNet-Mono) when fusing segmentations compared to single-view segmentation. This performance gain is observed in the qualitative results in Figure 7.4. It can be seen that our multi-view consistency training and Bayesian fusion produces more accurate and homogeneous segmentations. Figure 7.5 shows typical challenging cases for our model.

We also compare classwise and average IoU scores for 13-class semantic segmentation on NYUDv2 in Table 7.3. The results of **Eigen and Fergus** (2015) are from their publicly available model tested on 320×240 resolution. The results demonstrate that our approach gives high performance gains across all occurrence frequencies of the classes in the dataset.

7.6 Conclusion

In this paper we propose methods for enforcing multi-view consistency during the training of CNN models for semantic RGB-D image segmentation. We base our CNN design on FuseNet (**Hazirbas et al.** 2017), a recently proposed CNN architecture in an encoder-decoder scheme for semantic segmentation of RGB-D images. We augment the network with multi-scale loss supervision to improve its performance. We present and evaluate three different approaches for multi-view consistency training. Our methods use an RGB-D SLAM trajectory estimate to warp semantic segmentations or feature maps from one view point to another. Multi-view max-pooling of feature maps overall provides the best performance gains in single-view segmentation and fusion of multiple views.

We demonstrate the superior performance of multi-view consistency training and Bayesian fusion on the NYUDv2 13-class and 40-class semantic segmentation benchmark. All multi-view consistency training approaches outperform single-view trained baselines. They are key to boosting segmentation performance when fusing network predictions from multiple view points during testing. On NYUDv2, our model sets a new state-of-the-art performance using an end-to-end trained network for single-view predictions as well as multi-view fused semantic segmentation without further post-processing stages such as dense CRFs. In future work, we want to further investigate integration of our approach in a semantic SLAM system, for example, through coupling of pose tracking and SLAM with our semantic predictions.

CHAPTER 8

Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform

ABSTRACT Dense pixelwise prediction such as semantic segmentation is an up-to-date challenge for deep convolutional neural networks (CNNs). Many state-of-the-art approaches either tackle the loss of high-resolution information due to pooling in the encoder stage, or use dilated convolutions or high-resolution lanes to maintain detailed feature maps and predictions. Motivated by the structural analogy between multi-resolution wavelet analysis and the pooling/unpooling layers of CNNs, we introduce discrete wavelet transform (DWT) into the CNN encoder-decoder architecture and propose WCNN. The high-frequency wavelet coefficients are computed at encoder, which are later used at the decoder to unpooled jointly with coarse-resolution feature maps through the inverse DWT. The DWT/iDWT is further used to develop two wavelet pyramids to capture the global context, where the multi-resolution DWT is applied to successively reduce the spatial resolution and increase the receptive field. Experiment with the Cityscape dataset, the proposed WCNNs are computationally efficient and yield improvements the accuracy for high-resolution dense pixelwise prediction.

8.1 Introduction

Dense pixelwise prediction tasks such as semantic segmentation, optical flow or depth estimation remain up-to-date challenges in computer vision. They find rapidly rising interests for applications such as autonomous driving, robotic vision and image scene understanding. Succeeded by its remarkable success in image recognition by **Krizhevsky et al.** (2012), deep convolutional neural networks (CNNs) have achieved state-of-the-art performances in dense prediction tasks such as semantic segmentation (**G. Lin et al.** 2017; **Pohlen et al.** 2017; **Zhao et al.** 2017) or single-image depth estimation (**Laina et al.** 2016).

Many dense prediction tasks consist of two concurrent goals: classification and localization.

Classification is well tackled by an end-to-end trainable CNN architecture, *e.g.*, VGGNet by **Simonyan and Zisserman** (2015) and ResNet by **K. He et al.** (2016a), which typically stacks multiple layers of successive convolution, nonlinear activation, and pooling. A typical pooling step, which performs either a subsampling or some strided averaging on an input volume, is favorable for the invariance of prediction results to small spatial translations in the input data as well as for the boost of computational efficiency via dimension reduction. Its downside, however, is the loss of resolution in output feature maps, which renders high-quality pixelwise prediction challenging.

Several remedies for such a dilemma have been proposed in the literatures. As suggested in **Badrinarayanan et al.** (2017); **Noh et al.** (2015), one may mirror the encoder network by a decoder network. Each upsampling (or unpooling) layer in the decoder network is introduced in symmetry to a corresponding pooling layer in the encoder network, and then followed by trainable convolutional layers. Alternatively, one may use dilated (also known as atrous) convolutions in a CNN encoder as proposed in **L.-C. Chen et al.** (2018); **L.-C. Chen et al.** (2015); **Yu and Koltun** (2016). This enables the CNN to expand the receptive fields of pixels as convolutional layers stack up without losing resolution in the feature maps, however, at the cost of significant computational time and memory. Another alternative is to combine a CNN low-resolution classifier with a conditional random field (CRF) **Krähenbühl and Koltun** (2011); **Krähenbühl and Koltun** (2013), either as a stand-alone post-processing step **L.-C. Chen et al.** (2018); **L.-C. Chen et al.** (2015) or combined with a CNN in an end-to-end trainable architecture **G. Lin et al.** (2016b); **Zheng et al.** (2015). The latter also comes with an increased demand in run-time for training and inference.

Motivated by close analogy between pooling (resp. unpooling) in an encoder-decoder CNN and decomposition (resp. reconstruction) in multi-resolution wavelet analysis, this paper proposes a new class of CNNs with wavelet unpooling and wavelet pyramid. We name the network WCNN. The first contribution with WCNN is to achieve unpooling with the inverse discrete wavelet transform (iDWT). To this end, DWT is applied at the encoder to decompose feature maps into frequency bands. The high frequency components are skip-connected to the decoder to perform iDWT jointly with the coarse-resolution feature maps. The wavelet unpooling does not require any additional parameters over baseline CNNs, where the overhead only comes from the memory to cache frequency coefficients from encoder. The second contribution of WCNN are two wavelet-based pyramid variants to bridge the standard encoder and decoder. The wavelet pyramids obtain global context from a receptive field of the entire image by exploiting multi-resolution DWT/iDWT. The experiments over the dataset Cityscape show that the proposed WCNN yields systematically improvements in dense prediction accuracy.

8.2 Related Work

Many challenging tasks in computer vision such as single image depth prediction or semantic image segmentation require models for dense prediction, since they either involve regressing quantities pixelwise or classifying the pixels. Most current state-of-the-art methods for dense prediction tasks are based on end-to-end trainable deep learning architectures. Early methods segment the image into regions such as superpixels in a bottom-up fashion. Predictions for the regions are determined based on deep neural network features (**Farabet et al.** 2013; **F. Liu et al.** 2015; **Yan et al.** 2015). The use of image-based bottom-up regions supports adherence of the dense predictions to the boundaries in the image.

Long et al. (2015) propose a FCN architecture for semantic image segmentation which successively convolves and pools feature maps of an increasing number of feature channels. The semantic segmentation is predicted on an intermediate lower resolution within the network. Feature maps at this and lower resolutions are concatenated and further classified to form the final prediction. This design allows for the use of pretrained CNNs that are trained on other tasks such as large-scale object recognition **Krizhevsky et al.** (2012). Since the introduction of FCNs, many variants are proposed, which are trained end-to-end and outperform the early two-stage methods. **Hariharan et al.** (2015) classify pixels based on feature vectors that are extracted at corresponding locations across all feature maps in a CNN. This way, the method combines features across all layers available in the network, capturing high-resolution detail as well as context in large receptive fields. However, this approach becomes inefficient in deep architectures with many wide layers. **Noh et al.** (2015) and **Dosovitskiy et al.** (2015a) propose encoder-decoder CNN architectures which successively unpool and convolve the lowest resolution feature map of the encoder back to a high output resolution. Since the feature maps in the encoder lose spatial information through pooling, **Noh et al.** (2015) exploint the memorized unpooling (**M. D. Zeiler et al.** 2011) to upscale coarse feature maps at the decoder stage, where the pooling locations are used to unpool accordingly. The FCN of **Laina et al.** (2016) uses the deep residual network (**K. He et al.** 2016a) as an encoder, where most pooling layers are replaced by stride-two convolution. For upscaling, the upprojection block is developed as an efficient implementation of upconvolution. The principle of upconvolution is developed by **Dosovitskiy et al.** (2015b), which first unpools a feature map by putting activations to one entry of a 2×2 block and then filter the sparse feature map with convolution. Details in the predictions of such encoder-decoder FCNs can be improved by feeding the feature maps in each scale of the encoder to the corresponding scale of the decoder (skip connections, *e.g.,* **Dosovitskiy et al.** (2015a)). In RefineNet by **G. Lin et al.** (2017), the decoder feature maps are successively refined using multi-resolution fusion with their higher resolution counterparts in the encoder. In this paper, we also reincorporate the high-frequency information that is discarded during pooling to successively refine feature maps in the decoder.

Some FCN architectures use dilated convolutions in order to increase receptive field without pooling and maintain high-resolution of the feature maps (**L.-C. Chen et al.** 2018; **L.-C. Chen et al.** 2015; **Yu and Koltun** 2016). These dilated CNNs trade high-resolution output with the high memory consumption, which quickly become a bottleneck for training with large batch size for encoder-decoder CNNs. The full-resolution residual network (FRRN) by **Pohlen et al.** (2017) is an alternative model which keeps features in a high-resolution lane and at the same time, extracts low-resolution higher-order features in an encoder-decoder architecture. The high-resolution features are successively refined from residuals computed through the encoder-decoder lane. While the model is highly demanding in memory and training time, it achieves high-resolution predictions that well adhere to segment boundaries. **Ghiasi and Fowlkes** (2016) take inspiration from Laplace image decompositions for their network design. They successively refine the high-frequency parts of the score maps in order to improve predictions at segment boundaries. Structured prediction approaches integrate inference in CRFs with deep neural networks in end-to-end trainable models (**Chandra and Kokkinos** 2016; **G. Lin et al.** 2016b; **Z. Liu et al.** 2015; **Zheng et al.** 2015). While the models are capable of recovering high-resolution predictions, inference and learning typically requires tedious iterative procedures. In contrast to those approaches, we aim to provide detailed predictions in a swift and direct forward pass. Recently, the pyramid scene parsing network (PSPNet) from **Zhao et al.** (2017) extracts global context features using a pyramid pooling module, which shows the benefit of aggregation global information for dense predictions. The pyramid design in PSPNet relies multiple average pooling layers with heuristic window size. In this work, we also propose a more efficient pyramid pooling stage based on multi-resolution DWT.

8.3 WCNN Encoder-Decoder Architectures

Recently, CNNs have demonstrated impressive performance on many dense pixelwise prediction tasks, including image semantic segmentation, optical flow estimation, and depth regression. CNNs extract image features through successive layers of convolution and non-linear activation. In encoder architectures, as the stack of layers gets deeper, the dimension of the feature vectors increases while the spatial resolution is reduced. For dense prediction tasks, CNNs with encoder-decoder architecture are widely applied in which the feature maps of the encoder are successively unpooled and deconvolved. Research on architectures for the encoder part is relatively mature, *e.g.,* the state-of-the-art CNNs such as VGGNet (**Simonyan and Zisserman** 2015) and ResNet (**K. He et al.** 2016a) are commonly used in various applications. In contrast, the design of the decoder has not yet converged to a universally accepted solution. While it is easy to reduce spatial dimension by either pooling or strided convolution, recovering a detailed prediction from a coarse and high-dimensional feature space is less straight-forward. In this paper, we make an analogy between CNN encoder-decoders to the multi-resolution wavelet transform (see Figure 8.1). We match the pooling operations of the CNN encoder with the multilevel forward transformation of a signal by a wavelet. The decoder performs the corresponding inverse wavelet transform for unpooling. The analogy is straight-forward: the wavelet transform successively filters the signal into frequency subbands while reducing the spatial resolution. The inverse wavelet transform successively composes the frequency subband back to full resolution. While the encoder and the decoder transform between different domains (*e.g.,* image-to-semantic segmentation versus image-to-image in wavelet transforms), we find that wavelet unpooling provides an elegant mechanism to transmit high-frequency information from the image domain to the semantic segmentation. It also imposes a strong architectural regularization, as the feature dimensions between the encoder and the decoder need to match through the wavelet coefficients.

8.3.1 Discrete Wavelet Transform

We briefly introduce main concepts of DWT (see **Mallat** (2009) for a comprehensive introduction). The multi-resolution wavelet transform provides localized time-frequency analysis of signals and images. Consider a 2D input data $X \in \mathbb{R}^{2M\times 2N}$, $\phi \in \mathbb{R}^2$ and $\psi \in \mathbb{R}^2$ as 1D low-pass and high-pass filters, respectively. Denote the indexed array element by x_{ij}, the single-level DWT is defined as follows,

$$\begin{aligned} y^{ll}_{kl} &= \sum_l \sum_k x_{2i+k,2j+l}\phi_k\phi_l, \quad & y^{lh}_{kl} &= \sum_l \sum_k x_{2i+k,2j+l}\phi_k\psi_l, \\ y^{hl}_{kl} &= \sum_l \sum_k x_{2i+k,2j+l}\psi_k\phi_l, \quad & y^{hh}_{kl} &= \sum_l \sum_k x_{2i+k,2j+l}\psi_k\psi_l. \end{aligned} \tag{8.1}$$

All the convolutions above are performed with stride 2, yielding a subsampling of factor 2 along each spatial dimension. Let the low-low frequency component $Y^{ll} := \{y^{ll}_{kl}\}$, the low-high frequency component $Y^{lh} := \{y^{lh}_{kl}\}$, the high-low frequency component $Y^{hl} := \{y^{hl}_{i,j}\}$, and the high-high frequency component $Y^{hh} := \{y^{hh}_{i,j}\}$. The DWT results in $\{Y^{ll}, Y^{lh}, Y^{hl}, Y^{hh}\} \in \mathbb{R}^{M\times N}$. Conversely, supplied with the wavelet coefficients, and provided that $\{\phi, \psi\}$ and $\{\tilde{\phi}, \tilde{\psi}\}$ are bi-orthogonal wavelet filters, the original input X can be reconstructed by the inverse DWT as

$$x_{ij} = \sum_l \sum_k y^{ll}_{kl}\tilde{\phi}_{i-2k}\tilde{\phi}_{j-2l} + y^{lh}_{kl}\tilde{\phi}_{i-2k}\tilde{\psi}_{j-2l} + y^{hl}_{kl}\tilde{\psi}_{i-2k}\tilde{\phi}_{j-2l} + y^{hh}_{kl}\tilde{\psi}_{i-2k}\tilde{\psi}_{j-2l}\,. \tag{8.2}$$

A cascaded wavelet decomposition successively performs Equation (8.1) on low-low frequency coefficients $\{(\cdot)^{ll}\}$ from fine to coarse resolution, while the reconstruction works reversely from coarse to fine resolution. In this sense, decomposition-reconstruction in multi-resolution wavelet analysis is in analogy to the pooling-unpooling steps in an encoder-decoder CNN (*e.g.*, **Noh et al.** (2015)). Moreover, it is worth noting that, while the low-frequency coefficients $\{(\cdot)^{ll}\}$ store local averages of the input data, its high-frequency counterparts, namely $\{(\cdot)^{lh}\}$, $\{(\cdot)^{hl}\}$, and $\{(\cdot)^{hh}\}$ encode local textures which are vital in recovering sharp boundaries. This motivates us to make use of the high-frequency wavelet coefficients to improve the quality of unpooling during the decoder stage and, hence, improve the accuracy of CNN in pixelwise prediction.

Throughout this paper, we extensively use the Haar wavelet for its simplicity and effectiveness to boost the performances of the underlying CNN. In this scenario, the Haar filters used for decomposition, see Equation (8.1), are given by

$$\phi = \left(\frac{1}{2}, \frac{1}{2}\right) \quad , \quad \psi = \left(\frac{1}{2}, -\frac{1}{2}\right) \,. \tag{8.3}$$

The corresponding reconstruction filters in Equation (8.2) are given by $\widetilde{\phi} = 2\phi$, $\widetilde{\psi} = 2\psi$, and hence the inverse transform reduces to a sum of Kronecker products (denoted with $\otimes$)

$$X = Y^{ll} \otimes \widetilde{\phi}^{\top} \otimes \widetilde{\phi} + Y^{lh} \otimes \widetilde{\phi}^{\top} \otimes \widetilde{\psi} + Y^{hl} \otimes \widetilde{\psi}^{\top} \otimes \widetilde{\phi} + Y^{hh} \otimes \widetilde{\psi}^{\top} \otimes \widetilde{\psi} \,. \tag{8.4}$$

With CNNs, data at every layer are structured into 4D tensors, *i.e.*, along the dimensions of the batch size, the channel number, the width and the height. To perform the wavelet transform for CNNs, we apply DWT/iDWT channelwise. Without confusion, the remaining text adopts the shorthand notations $G_h(X)$ for the Haar DWT and $G_h^{-1}(Y^{ll}, Y^{lh}, Y^{hl}, Y^{hh})$ for the corresponding iDWT.

8.3.2 Wavelet CNN Encoder-Decoder Architecture

We propose a CNN encoder-decoder that resembles multi-resolution wavelet decomposition and reconstruction by its pooling and unpooling operations. In addition, we introduce two pyramid variants to capture global contextual features based on the wavelet transformation.

Figure 8.1 gives an overview of the proposed WCNN architecture. WCNN employs ResNet developed by **K. He et al.** (2016a) for the encoder. In ResNet, the input resolution is successively reduced by a factor of 32 via one max-pooling layer and four stride-two convolutional layers, *i.e.*, conv1, conv3_1, conv4_1 and conv5_1. In order to restore the input resolution with the decoder, WCNN inserts three DWT layer after conv2, conv3 and conv4 to decompose the corresponding feature maps into four frequency bands. The high frequencies Y^{lh}, Y^{hl}, Y^{hh} are skip-connected to the decoder to perform unpooling via the iDWT layers, which we will discuss in details with Section 8.3.2.1. We add three convolutional residual block **K. He et al.** (2016a) to filter the unpooled feature maps further before the next unpooling stage. As illustrated in Figure 8.1, the three iDWT layers upsample the output to 1/4 input resolution. The full-resolution output is obtained with two upconvolutional blocks by transposed convolution. To bridge the encoder and decoder, the contextual pyramid with wavelet transformation is added. Section 8.3.2.2 will detail the pyramid design.

8.3.2.1 Wavelet Unpooling

WCNN achieves the unpooling through iDWT layers. To this end, the DWT layers are added consistently into the encoder to obtain high-frequency components. The idea is straightforward. At encoder, the DWT layers decompose the feature map into four frequency bands channelwise, where each frequency band is half-resolution of the input. The high-frequency components are skip-connected to the decoder where the spatial resolution needs to be upscaled by a factor of two. Taking the layer idwt_4 in Figure 8.1 as an example, the input to this layer are four components of spatial resolution 1/32 to perform iDWT. The pyramid output serve the low-low frequency $\widetilde{Y}^{ll}$, while the output of the dwt4 layer operating on the conv4 provide the three high-frequency components Y^{lh}, Y^{hl}, and Y^{hh}. With iDWT, the spatial resolution is upscaled to 1/16. The output of layer idwt4 is finalized by adding the 1/16 resolution direct output of conv4, which is a standard skip connection commonly used

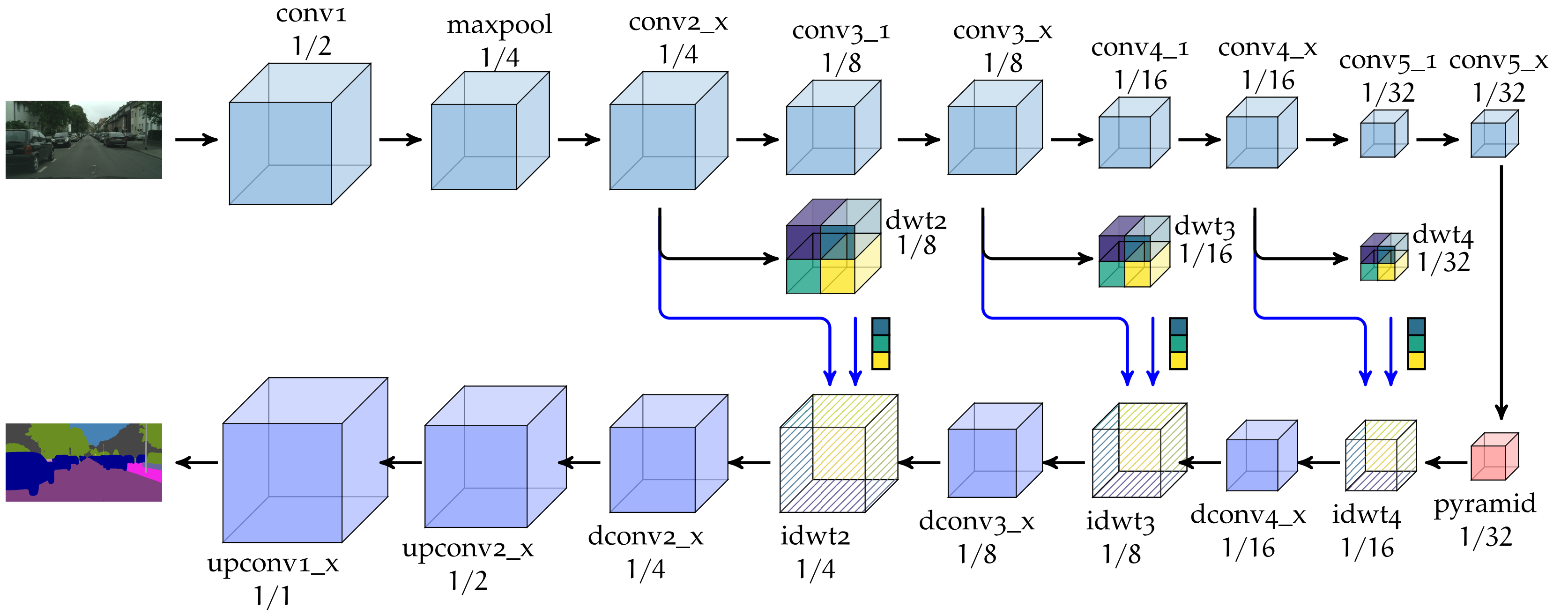
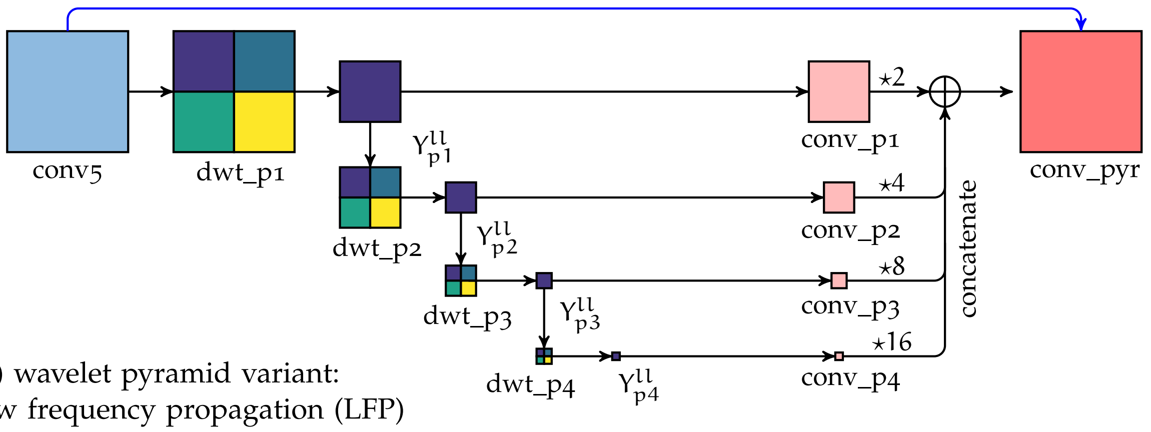
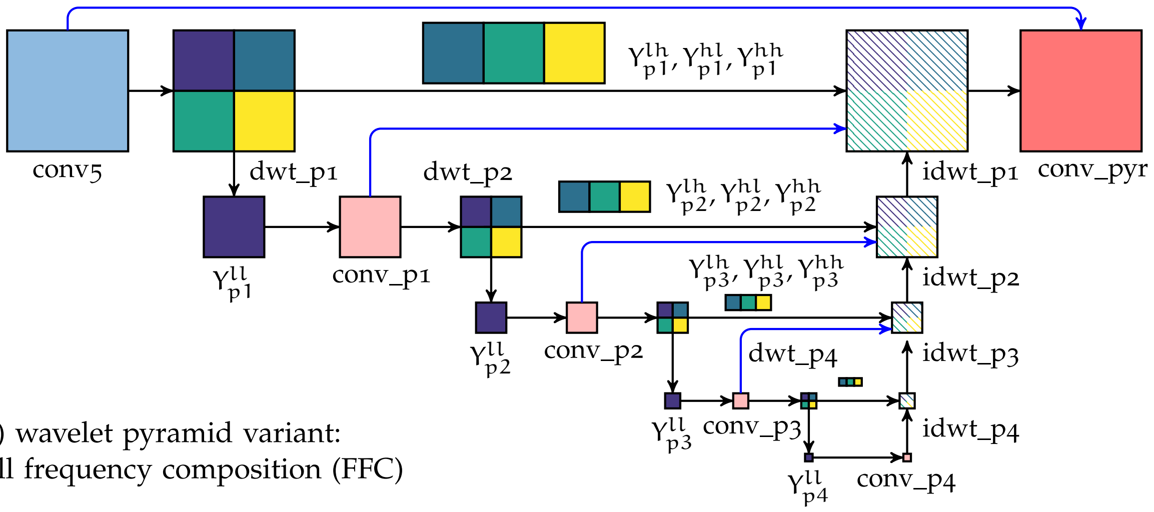


Figure 8.1: The encoder-decoder architecture of the proposed WCNN, where the data flow is indicated by black arrows and shortcuts are by blue arrows. Assume the input resolution is 1, the output resolution of each building block is denoted by 1/x. WCNN employs ResNet (**K. He et al.** 2016a) for the encoder, which reduces the input resolution by a factor of 32 via stride-two convolutional layers, except for one maxpool layer after conv1. To restore the input resolution, WCNN inserts three DWT layer after conv2, conv3 and conv4. The high frequencies from DWT layers are used in the decoder to perform unpooling by the iDWT layers. To extract global context, WCNN introduces two pyramid variants to bridge the encoder and decoder, which also exploits DWT/iDWT layers (see details in Figure 8.2).



(a) wavelet pyramid variant:
low frequency propagation (LFP)



(b) wavelet pyramid variant:
full frequency composition (FFC)

Figure 8.2: The proposed wavelet pyramid variants, with the data flow indicated by black arrows and shortcuts by blue arrows. Both pyramids take conv5 as input and produce conv_pyr as output, without changing the spatial resolution. Both pyramids build a multi-resolution wavelet pyramid via successive DWT. The LFP pyramid only utilizes the low-low frequency Y^{ll}, where each Y^{ll} is filtered by further convolutions, bilinear upsampled to the input resolution and concatenated. The FFC pyramid employs the high frequency bands for upscaling via iDWT.

by many state-of-the-art encoder-decoder CNNs to improve the upsampling performance. The iDWT layer can thus be described by

$$G_h^{-1}(\widetilde{Y}^{ll}, Y^{lh}, Y^{hl}, Y^{hh}) + X \,. \tag{8.5}$$

We denote this appproach of upscaling the decoder feature map with the wavelet coefficients from the encoder as wavelet unpooling.

Typically, CNNs extract feature with many layers of convolution and nonlinear operations, which transform and embed the feature space differently layer by layer. The wavelet unpooling aims to maintain the similar frequency structure throughout CNNs. By replacing the low-frequency of the encoder with the corresponding output of the decoder to perform iDWT with the high-frequency bands from the encoder, the wavelet unpooling aims to enforce learning feature maps of invariant frequency structure under layers of filtering. The skip connections of the signals before DWT also support learning such consistency.

In comparison to the other unpooling methods, for example to upsampling by transposed convolution as proposed in **Long et al.** (2015), wavelet unpooling does not require any parameters for both DWT and iDWT layers. Compare to the memorized unpooling as proposed in **M. D. Zeiler et al.** (2011), or the method to map the low-resolution feature map to the top-left entry of a 2×2 block (**Dosovitskiy et al.** 2015b), the wavelet unpooling aims to restore every entries according to the frequency structure.

8.3.2.2 Wavelet Pyramid

With CNNs that are designed for classification task, the last few layers typically reduce the spatial resolution to 1×1. Such feature maps have the receptive field of the entire input image and therefore capture the global context. Recent works have demonstrated that capturing global context information is also crucial for accurate dense pixelwise prediction (**L.-C. Chen et al.** 2015; **Zhao et al.** 2017). While it is straight-forward to obtain global context with fully-connected layer or with convolutional layers of large filter size, it is difficult to bridge an encoder with drastically reduced spatial resolution to a proper decoder. Most state-of-the-art CNN encoder reduce the spatial resolution by a factor of 32, which produces 7×7 output given 224×224 input dimensions. If the global context is captured by a simple fully-connected layer, learning 7×7 upsampling kernels is challenging.

One solution is to use the dilated convolutions, which increase the perceptive field with the same amount of parameters (**L.-C. Chen et al.** 2018; **L.-C. Chen et al.** 2015). Building on the dilated CNNs, the pyramid spatial pooling network PSPNet by **Zhao et al.** (2017) introduces a pyramid on the feature map with multiple average pooling of different window sizes. Noticeably, the dilated convolutions demand considerably larger amounts of memory to host the data, which quickly becomes the bottleneck for training with large batch size. In this work, we base our network design on non-dilated CNNs and instead construct the pyramids through wavelet transformations. We propose two wavelet pyramids variants, namely the low frequency propagation (LFP) and the full frequency composition (FFC) as shown in Figure 8.2.

Low-Frequency Propagation Wavelet Pyramid Shown in Figure 8.2 (a), the LFP pyramid successively performs DWT on the low-low frequency components Y^{ll}. At each pyramid level, the extracted Y^{ll} component is further transformed with a convolutional layer, which is then bilinear upsampled to the same spatial resolution as the pyramid input, *i.e.,* conv5. We then concatenate these the upsampled feature maps to aggregate the global context that are captured at different scale. This concatenated feature map is combined with the skip-connected conv5 by an elementwise addition, which sis then filtered with a 1×1 convolutional layer to match the channel dimension of the decoder.

With LFP, a multi-resolution wavelet pyramid is constructed, where only the low-low frequency bands of each level are used. The LFP pyramid resembles the pyramid proposed by the PSPNet (**Zhao et al.** 2017). In particular, with the Haar wavelet, the low-low frequency is equivalent to the average pooling by a 2×2 window. However, the difference is the PSPNet design average pooling with a multiple heuristic window size, whereas LFP pyramid is strictly performed accordingly to frequency decomposition. Despite the Haar wavelet is used in this work, the LFP pyramid can be easily generalized with other wavelet base functions.

Full-Frequency Composition Wavelet Pyramid The LFP pyramid only uses the low-low frequency bands. In order to make full use of the frequency decomposition, the FFC pyramid is developed. Shown in Figure 8.2 (b), the FFC pyramid amounts to a small encoder-decoder with wavelet unpooling. Start from the input conv5, DWT is performed to obtain the four frequency bands. The low-low frequency band Y^{ll} is filtered by an additional convolutional layer and the high frequency bands Y^{lh}, Y^{hl}, Y^{hh} are cached for unpooling. The filtered low-low frequency is then further decomposed by DWT into the finer level and the same operation repeats until the finest feature map is obtained. To upscale from the finest level, we again adopt the wavelet unpooling as described by Equation (8.5). To this end, the iDWT is first performed using the cached high frequency bands, and then the output is further fused with the skip connection. The wavelet unpooling successively restore the spatial resolution to the same as the input to the pyramid. Finally, we skip connect conv5 with the wavelet output by an elementwise addition, and project the global context with a 1×1 convolution to bridge the following decoder. It can be seen that, the FFC pyramid mimic the encoder-decoder design, which naturally reduces the spatial resolution and restore it in the consistent manner with the remaining network.

8.4 Evaluations

In this section, we evaluate the proposed WCNN method for the task of semantic image segmentation. To this end, we use the Cityscape benchmark dataset **Cordts et al.** (2016) which contains 2,975 training, 500 validation and 1,525 test images that are captured in 50 different cities from a driving car. All the images are densely annotated into 30 commonly observed objects classes occurring in urban street scenes from which 19 classes are used for evaluation. The Cityscape benchmark provides all the images with the same high resolution of 2048×1024. The ground truth for the test images is not publicly available and evaluations on the test set are submitted online[1] for fair comparison.

[1] http://www.cityscapes-dataset.com

8.4.1 WCNN Configurations

Table 8.1 presents the network configurations of the proposed WCNN. We take the state-of-the-art ResNet101 **K. He et al.** (2016a) for the encoder. The ResNet101 uses stride-two convolution to reduce spatial resolution. To implement WCNN, we preserve the stride-two convolution layers and insert three DWT layers dwt2, dwt3, dwt4 into the decoder conv2_x, conv3_x, conv4_x, respectively to obtain the frequency bands. At each upscaling stage at the decoder, the corresponding frequency bands are used, then followed by several residual blocks before the next upscaling stage. The last two upscaling stages are implemented as upconvolution, where transposed convolution is first applied to scale up the resolution by a factor of two, then residual blocks are used to further filter the intermediate output. In WCNN, we reply heavily on the residual blocks proposed in ResNet **K. He et al.** (2016a), where each block is a stack of three convolutional layers with the second layer of 3×3 for feature extraction and the first and third layers as 1×1 convolutions for feature projection.

In this work, we develop CNNs for high-resolution predictions. An input image of 512×1024 yields conv5_x to have the spatial resolution of 16×32. Therefore, we design both LFP and FFC pyamids to have four levels of DWT, which produce the four levels of frequency components of 8×16, 4×8, 2×4 and 1×2, respectively. The finest pyramid level thus has the receptive field of the entire input. The details of the LFP and FFC pyramids are given in Table 8.2.

To evaluate the proposed network, the baseline CNN is designed to have minimum difference with WCNN. Taking the WCNN configuration in Table 8.1, the baseline model 1) removes all DWT layers at encoder 2) replaces the pyramid by one $1 \times 1, 1024$ convolutional layer, and 3) replaces the iDWT layers by transposed convolution to upscale the feature map by a factor of 2. The rest layers are the same with WCNN. In the following experiment, we compare the baseline model, the baseline model with LFP and FFC pyramid, the WCNN with LFP and FFC pyramids.

8.4.2 Implementation Details

We have implemented all our methods based on the TensorFlow (**Abadi et al.** 2016) machine learning framework. For network training, we initialize the parameters of the encoder layers from pretrained ResNet model on ImageNet and initialize the convolutional kernels on the decoder with the initialization methods proposed by **K. He et al.** (2015). We run the training with batch size of four on the Nvidia Titan X GPU. For both training, we minimize the cross-entropy loss using the Stochastic Gradient Descent (SGD) solver with Momentum of 0.9. The initial learning rate is set to 0.001 and decrease with a factor of 0.9 every 10 epoch. We train the network until convergences. For cityscapes, all the variants of our experiments converges around 60K iterations. Following **Pohlen et al.** (2017), we apply bootstrapping loss minimization for Cityscapes benchmark in order to speed up the training and boost the segmentation accuracy. For all Cityscapes experiments, we fix the threshold of bootstrapping to the top 8192 most difficult pixels per images.

Table 8.1: The layer configurations of the proposed WCNN (see Figure 8.1). The encoder is based on ResNet101 **K. He et al.** (2016a). The resblock is the residual block from ResNet, where $(x, y) \times z$ denotes stacking z blocks of $[(1 \times 1, x), (3 \times 3, x), (1 \times 1, y)]$ convolutional layers. For upconvolution, the transposed convolution is first used to upscale the input by a factor of two, followed by residual blocks. We denote the stride-two operations with s2, and elementwise addition with $\boxplus$. The dimension of the layer output assumes the spatial resolution of input image is normalized to 1, and the second entry denotes the depth of the feature maps.

	layer	operation	input	dimension
ResNet101-based Encoder	conv1	$(7 \times 7, 64)$, s2	RGB	1/2, 64
	maxpool	(2×2), s2	conv1	1/4, 64
	conv2_x	resblock $(64, 256) \times 3$	maxpool	1/4, 256
	dwt2	G_h	conv2_x	1/8, 256
	conv3_1	resblock $(128, 512)$, s2	conv2_x	1/8, 512
	conv3_x	resblock $(128, 512) \times 3$	conv3_1	1/8, 512
	dwt3	G_h	conv3_x	1/16, 512
	conv4_1	resblock $(256, 1024)$, s2	conv3_x	1/16, 1024
	conv4_x	resblock $(256, 1024) \times 22$	conv4_1	1/16, 1024
	dwt4	G_h	conv4_x	1/32, 1024
	conv5_1	resblock $(512, 2048)$, s2	conv4_x	1/32, 2048
	conv5_x	resblock $(512, 2048) \times 2$	conv5_1	1/32, 2048
	pyramid		conv5x	1/32, 1024
WCNN Decoder	idwt4	G_h^{-1}	pyramid, $Y_4^{lh}, Y_4^{hl}, Y_4^{hh}$	1/16, 1024
	dconv4_x	resblock $(256, 512) \times 3$	idwt4 $\boxplus$ conv4_x	1/16, 512
	idwt3	G_h^{-1}	dconv4_x, $Y_3^{lh}, Y_3^{hl}, Y_3^{hh}$	1/8, 512
	dconv3_x	resblock $(128, 256) \times 3$	idwt3 $\boxplus$ conv3_x	1/8, 256
	idwt2	G_h^{-1}	dconv3_x, $Y_2^{lh}, Y_2^{hl}, Y_2^{hh}$	1/4, 256
	dconv2_x	resblock $(64, 128) \times 3$	idwt2 $\boxplus$ conv2_x	1/4, 128
	upconv2_x	upconv $(64, 64) \times 3$	dconv2_x	1/2, 64
	upconv1_x	upconv $(64, 64) \times 2$	upconv2_x	1/1, 64

Table 8.2: The configurations of the proposed LFP and FFC pyramids (see Figure 8.2). Assuming conv5 has a resolution of $16 \times 32, 2048$, both LFP and FFC pyramids have four levels. For simplicity, the outer two levels are presented in the table, whereas the inner two levels repeats the same patterns. The operator $\star a$ denotes bilinear upsample by a factor of a and the operator $\boxplus$ denotes elementwise addition.

	layer	operation	input	dimension
LFP-pyramid	dwt_p1	G_h	conv5	8×16, 2048
	conv_p1	$(1 \times 1, 512)$	Y^{ll}_{p1}	8×16, 512
	dwt_p2	G_h	Y^{ll}_{p1}	4×8, 512
	conv_p2	$(1 \times 1, 512)$	Y^{ll}_{p2}	4×8, 512
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	concat	concatenation	$Y^{ll}_{p1} \star 2, Y^{ll}_{p2} \star 4 Y^{ll}_{p3} \star 8, Y^{ll}_{p4} \star 16$	16×32, 2048
	conv_pyr	$(1 \times 1, 1024)$	concat $\boxplus$ conv5	16×32, 1024
FFC-pyramid	dwt_p1	G_h	conv5	8×16, 2048
	conv_p1	$(1 \times 1, 2048)$	Y^{ll}_{p1}	8×16, 2048
	dwt_p2	G_h	conv_p1	4×8, 2048
	conv_p2	$(1 \times 1, 2048)$	Y^{ll}_{p2}	4×8, 2048
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	idwt_p2	G_h^{-1}	conv_p2$\boxplus$ idwt_p3, $Y^{lh}_2, Y^{hl}_2, Y^{hh}_2$	8×16, 2048
	idwt_p1	G_h^{-1}	conv_p1$\boxplus$ idwt_p2, $Y^{lh}_1, Y^{hl}_1, Y^{hh}_1$	16×32, 2048
	conv_pyr	$(1 \times 1, 1024)$	idwt_p1$\boxplus$ conv5	16×32, 1024

To train all the variants of the baseline and our model, we fix the input to the network to quarter resolution of the original dataset, *i.e.*, 512×1024. For evaluation on the validation dataset, we upsample the output logits bilinear to half of the resolution (to match the network input resolution) and compute the intersection-over-union (IoU) score for each class and on average. We also experiment with test time data augmentation, where we randomly scale the input images and feed them through the network before fuse the score.

8.4.3 Cityscapes

We evaluate segmentation accuracy using the commonly used evaluation metric of IoU. Table 8.3 gives the class-wise IoU and the mean IoU over the 19 classes. It can be seen that adding LFP and FFC pyramids to the baseline network already significantly improves the segmentation performance over the baseline. The FFC pyramid consistently outperforms the LFP pyramid. With WCNN we gain another increase in mean IoU of up to 1.2 over the corresponding baseline. With multi-scale test time augmentation, the accuracy of each model is increased, but the similar rank is observed among the different methods. Our variants strongly benefit, while the combination of wavelet unpooling and FFC wavelet pyra-

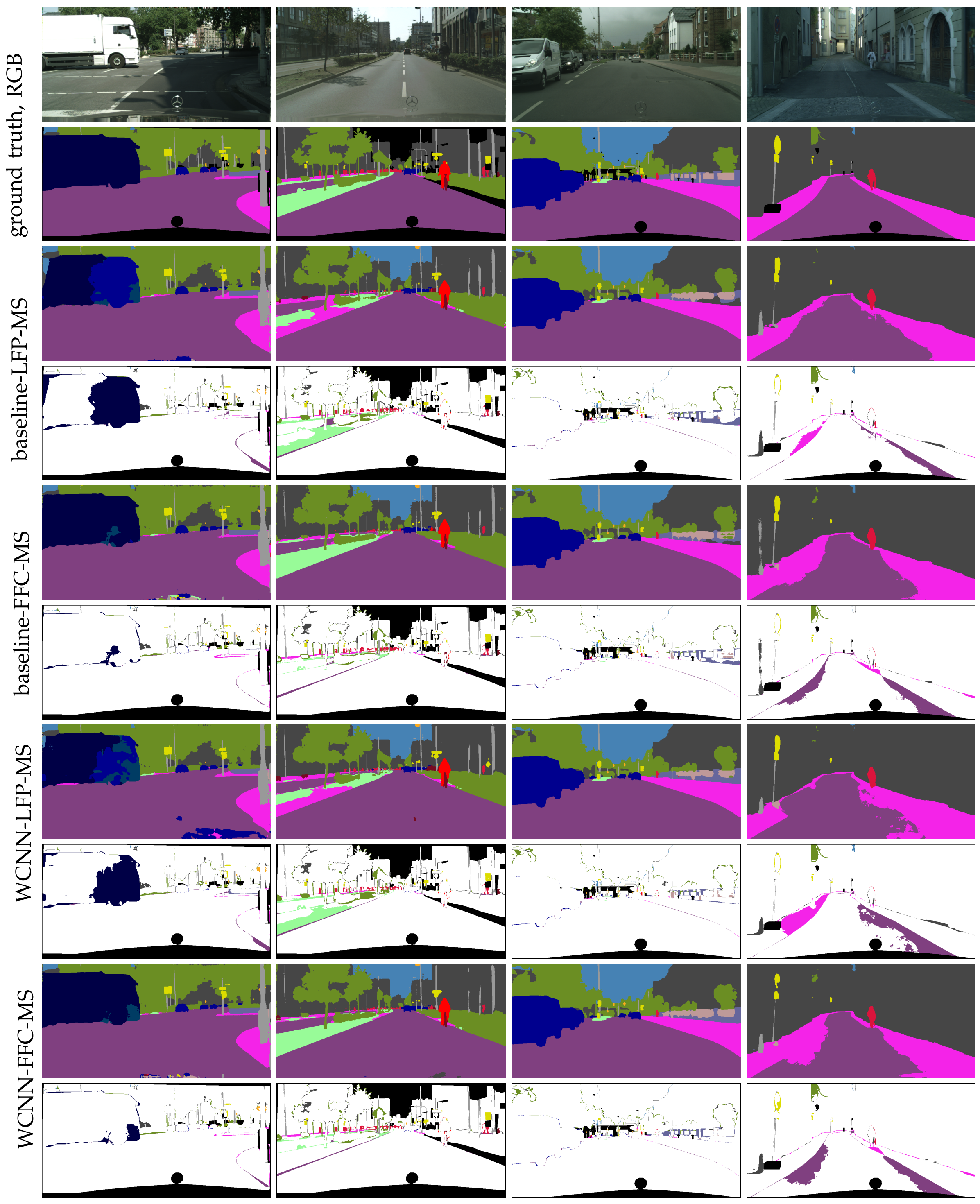


Figure 8.3: Qualitative exemplary semantic segmentation results on the Cityspaces dataset. From top to bottom: RGB image, ground-truth segmentation, baseline-LFP-MS, baseline-FFC-MS, WCNN-LFP-MS, WCNN-FFC-MS. The semantic color coding is given in Table 8.3.

Table 8.3: Cityscapes 19-class semantic segmentation IoU scores on *val* set. All test results are obtained by comparing to half resolution ground-truth labeling, which is the resolution of input images into our networks. The second part of the table report the performance with multi-scale test time data augmentation, indicated by the MS suffix.

method	■ road	■ sidewalk	■ building	■ wall	■ fence	■ pole	■ traffic	■ traffic light	■ vegetarian	■ terrain	■ sky	■ person	■ rider	■ car	■ truck	■ bus	■ train	■ motorcycle	■ bicycle	avg
frequency	37.7	5.4	21.9	0.7	0.8	1.5	0.2	0.7	17.2	0.8	3.4	1.3	0.2	6.6	0.3	0.4	0.1	0.1	0.7	
baseline	98.8	88.8	96.0	51.5	61.6	62.0	66.6	76.5	96.0	70.1	97.1	85.8	66.4	97.0	81.4	85.4	59.0	53.8	84.6	69.2
baseline-LFP	98.6	90.1	95.5	62.6	62.6	61.3	65.7	76.0	95.9	69.3	97.4	85.4	63.6	97.1	80.1	88.4	73.8	61.2	85.1	71.2
baseline-FFC	98.6	89.6	95.3	63.4	62.0	61.3	67.8	74.4	96.1	64.6	97.3	85.9	63.0	96.9	85.5	89.4	73.6	58.5	84.5	70.7
WCNN-LFP	98.6	89.8	95.7	63.0	65.8	61.5	67.8	76.2	96.3	69.4	97.4	85.8	67.4	97.2	82.0	88.9	69.9	59.9	84.9	71.6
WCNN-FFC	98.7	90.5	95.6	64.8	64.6	63.2	67.8	77.3	96.1	71.0	97.3	86.1	65.3	97.0	82.7	88.7	77.6	57.7	85.1	71.9
baseline	**99.0**	90.6	**96.7**	48.0	61.2	68.2	72.9	80.2	96.3	**72.5**	97.7	89.1	70.3	97.6	76.6	82.2	48.9	60.7	84.9	71.4
baseline-LFP-MS	98.7	92.2	96.5	54.0	65.5	68.9	71.2	79.0	96.1	64.7	97.6	88.1	64.3	**97.8**	71.2	87.3	71.8	**68.5**	85.7	73.3
baseline-FFC-MS	98.7	91.7	96.4	64.6	65.0	67.4	**74.3**	79.7	**96.7**	68.9	**98.0**	88.8	68.9	97.5	**88.3**	**90.6**	**79.3**	60.9	**85.8**	74.7
WCNN-LFP-MS	98.8	**92.4**	96.2	61.2	**68.0**	68.5	71.2	79.8	96.3	64.8	97.5	88.4	**70.1**	**97.8**	77.8	89.3	61.6	74.1	87.1	73.9
WCNN-FFC-MS	98.8	92.2	96.6	**68.6**	64.8	**69.1**	73.9	**81.6**	**96.7**	72.4	97.8	**89.3**	68.9	97.5	87.3	90.5	73.3	58.0	85.3	**75.2**

Table 8.4: IoU scores for the Cityscapes 19-class and category semantic segmentation on the *test* set (benchmark). All test results are obtained by testing on half resolution and comparing to full resolution groundtruth labeling through upsampling.

method	class mIoU	category mIoU
FRRN (**Pohlen et al.** 2017)	71.8	**88.9**
WCNN-FFC	70.9	86.1
WCNN-FFC-MS	**73.7**	88.3

mid achieves best increase in performance towards the baseline (6.0 mIoU). These results demonstrate that wavelet unpooling as well as the FFC wavelet pyramid improve the dense prediction of the baseline model. The qualitative comparisons are shown in Figure 8.3. It can be seen that the WCNN approach recovers fine-detailed structures such as fences, poles or traffic signs with higher accuracy than the baselines.

Table 8.4 compares our method with the current state-of-the-art method FRRN **Pohlen et al.** (2017) on the same input resolution (2x subsampling) on the Cityscapes benchmark. It can be seen that our method WCNN-FFC-MS outperforms FRRN by 1.9 mean IoU over the 19-classes while it is worse (0.6 mIoU) on the category level. Notably, WCNN is much less memory demanding than FRRN.

8.5 Conclusion

This paper introduce WCNN, a novel encoder-decoder CNN architecture for dense pixelwise prediction. The key innovation is to exploits the discrete wavelet transform (DWT) and inverse DWT to design the unpooling operation. In the proposed network, the high-frequency coefficients extracted by DWT at the encoder stage are cached and later combined with coarse-resolution feature maps at the decoder to perform accurate upsampling and hence, ultimate pixelwise prediction. Further, two wavelet pyramid variants are introduced, *i.e.*, the low frequency propagation (LFP) pyramid and the full frequency composition (FFC) pyramid. Both pyramid extract the global context from the encoder output with multi-resolution wavelet decomposition. Shown in experiment, WCNN outperforms the variant baseline CNNs and achieve the state-of-the-art semantic segmentation performance on the Cityscapes dataset.

In the future work, we will evaluate WCNNs for different dense pixelwise prediction tasks, *e.g.*, depth estimation and optical flow estimation. We will also perform ablation study of the wavelet pyramid to evaluate different pyramid configuration. It is also interesting to extend the WCNN for different wavelet base functions or ultimately learn the optimal base functions with CNNs.

CHAPTER 9

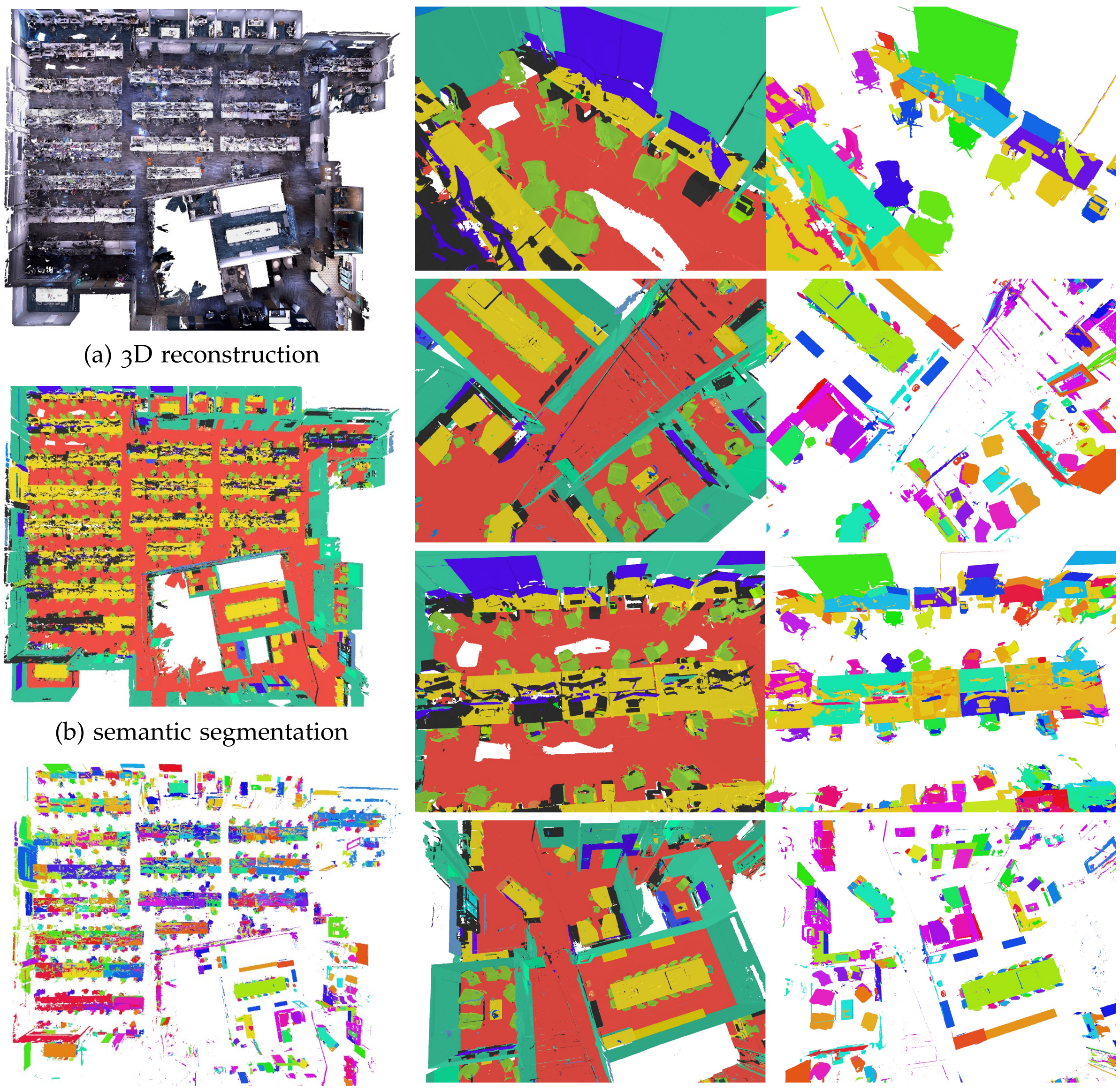
Human-in-the-loop Annotation for Large-scale 3D Semantic Datasets

ABSTRACT We propose a unified tool to generate high quality, geometrically consistent 3D semantic annotation for large scale static scenes. Our novel tool enables an individual or team to annotate large scale, fine grained meshes with class and instance labels. We show experimentally that the proposed tool produces a clean 3D segmentation and the geometric consistency of the labels enables automatic generation of 2D image segmentations, both of which are useful in many state-of-the-art ML based computer vision and AI tasks beyond the segmentation task itself including modern joint depth and semantic segmentation from passive image data and embodied question answering challenges.

9.1 Introduction

Semantic scene understanding is a fundamental problem in computer vision, robotics and graphics. To facilitate the research, large-scale high-quality ground-truth datasets are crucial to data-hungry algorithms, such as deep learning. Recently, there have been some releases of large-scale semantic RGB-D datasets. While these datasets have an impressive scale, little work focuses on how to efficiently obtain detailed annotations that also preserve consistency between multi-view 2D images and 3D reconstruction. In this paper, we develop a tool that not only enables humans to annotate directly on 3D meshes, but also enables clean up of mesh geometry to enhance semantic propagation between 2D and 3D. Aiming for large-scale ground truth collection, we propose a closed-loop workflow, that learns from existing semantic segmentations to bootstrap future annotations.

State-of-the-art annotation tools either label 2D images (**Russell et al.** 2008; **Silberman et al.** 2012) or operate on segmented 3D meshes (**Dai et al.** 2017a; **Nguyen et al.** 2017). Shown in Figure 9.2, the existing annotations are often contaminated with labeling noise, the correspondence between 2D and 3D labels are either absent or inconsistent, and it is difficult to


(a) 3D reconstruction

(b) semantic segmentation

(c) instance segmentation

(d) zoomed-in views of semantic and instance segmentation

🟥 floor	🟩 wall	🟨 table	🟩 chair	🟧 bin	🟦 ceiling	🟦 sofa	🟩 cabine
🟦 plant	🟩 door	🟪 shades	🟪 monitor	🟪 screen	🟨 blanket	🟥 window	⬛ unknown

Figure 9.1: The semantic instance segmentation of an approximately 2500m^2 office space, obtained by training deep convolutional neural network (Mask-RCNN by K. He et al. (2017)) from 20 indoor scenes annotated by our algorithm and fusing 2D predictions onto 3D mesh reconstruction. With the proposed pipeline, most surfaces are correctly initialized, which bootstraps human effort for further annotations.

obtain detailed annotations that respect object boundaries. Additionally, 3D datasets heavily rely on structured light depth sensor for dense reconstruction. Because of occlusions and lack of coverage and sensor limitations, missing surfaces are common in the reconstruction. This fact leads to erroneous data associations when projecting 3D to 2D image space and vice versa. This severely limits the use of semantic 3D datasets to train and benchmark 2D machine learning algorithms. With the aim of large-scale annotations, annotation efficiency is an important concern. Along those lines, we investigate whether existing annotations can assist further labeling. This work makes the following key contributions:

- A segmentation-aided free-form mesh labeling algorithm, which yields hierarchical annotation with better detail and efficiency.
- A human-aided geometry correction technique to insert missing surfaces to enhance 2D/3D association.
- A closed-loop bootstrapping annotation scheme, which trains instance semantic annotations from annotated data, and integrates the predictions back into a mesh segmentation.

To demonstrate the effectiveness of our method, we show experimentally that compared to a theoretically optimal accuracy of a segment-based annotation, we increases the accuracy by at least 5%. We demonstrate that the reconstruction amended by our method reduces rendering faults and produces better ground-truth annotation. In comparison to the state-of-the-art segmentation-based algorithm, our annotated model renders 2D label images that gives approximately 20% better accuracy. To demonstrate the close-loop annotation pipeline, we collect and annotated 20 indoor scenes with our method and used them to learn semantic instance prediction in 2D images with the state-of-the-art deep neural network Mask-RCNN by **K. He et al.** (2017). With the novel fusion method proposed in this work, we demonstrate that even with comparatively little data, our pipeline produces reasonable semantic and instance 3D mesh segmentation. Shown in Figure 9.1, the method automatically initialized 95% of the surface annotations of an entire office floor.

9.2 Related Works

RGB-D Annotation One early work for interactive 3D semantic annotation is developed by SemanticPaint by **Valentin et al.** (2015) to label indoor scene reconstruction and is extended to outdoor with passive stereo cameras by **Miksik et al.** (2015). The algorithm takes in RGB-D sequences, reconstruct the scene with KinectFusion (**Newcombe et al.** 2011a) and labels each voxels with limited human interaction, *i.e.*, briefly painting strokes on raycasted views. The authors then perform online learning with a random forest to train a semantic classifier based on a human annotation. The classifier provides the unary energy for a dynamic fully connected CRF (**Krähenbühl and Koltun** 2011) over the voxels, which enables label propagation from limited human annotation to the entire model.

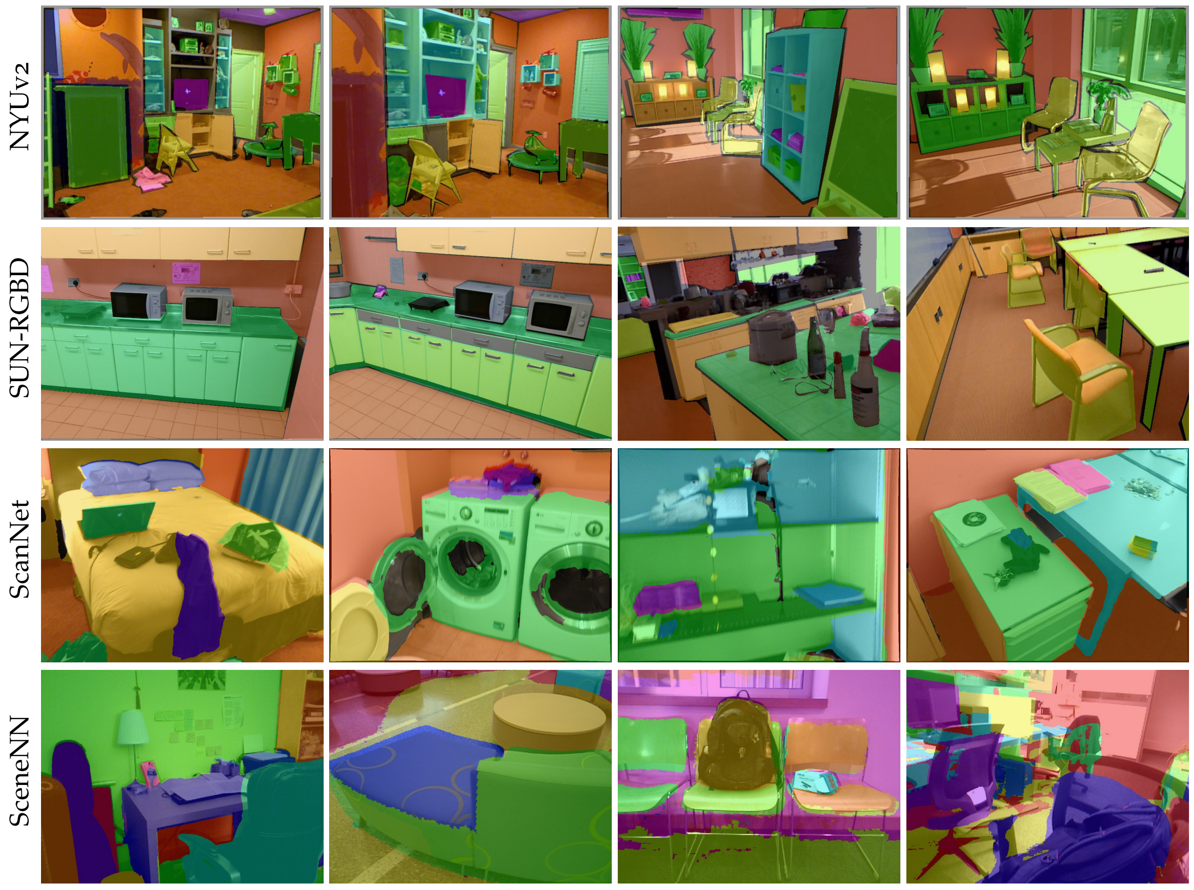


Figure 9.2: The limitations of state-of-art RGB-D annotations. NYUv2 (**Silberman et al.** 2012) and SUN-RGBD (**Song et al.** 2015) use polygon-based annotation, which leads to label inconsistency between multiple viewpoints. ScanNet (**Dai et al.** 2017a) and SceneNN (**Nguyen et al.** 2017) proposed segment-based mesh annotations, where objects are not accurately extracted and details are lost. In both scenarios, 2D labels are offset with respect to the actual object boundary.

In the effort to generate a large-scale semantic RGB-D dataset, many annotation strategies have been proposed. One solution is to ignore the depth and multi-view information and directly annotate in 2D. Based on drawing 2D polygons around objects, NYUv2 (**Silberman et al.** 2012) obtained a total amount of 1449 annotated images with semantic labels. The drawback of single-view annotation is inconsistency in labeling between different viewpoints and the inefficiency of annotating the same scene parts multiple times. To solve this problem, **Song et al.** (2015) propose the SUN-RGBD dataset, where 3D point cloud is annotated by drawing gravity-aligned 3D bounding box around object instances and projecting the result back onto 2D images. However, not all objects can be extracted by bounding box.

As a result, labels are still inconsistent and annotation by projecting 3D bounding box leads to labeling noise. Given RGB-D dataset are typically a sequence of video rather than single view images, **Hua et al.** (2016) propose to perform annotation on a reconstructed 3D model. The annotation is initialized with an over-segmentation of mesh, and human effort is dedicated to merge, extract and split segments to create annotations. This annotation pipeline is further developed by **Nguyen et al.** (2017), where repetitive objects are searched using a sliding window to obtain automatic label transfer. The authors additionally propose to project 3D labels to 2D images to generate fully annotated video sequences. To compensate projection misalignment, a bipartite graph matching method is applied for boundary refinement. Based on similar segmentation-based 3D mesh annotation method, **Dai et al.** (2017a) proposed a large-scale semantic dataset by crowd-sourcing annotation. One drawback of the segmentation-based labeling is that the error in segments, such as offset to object boundary, loss of instance details are propagated into the final annotation results. Recently, the Matterport cameras, which provide 360° RGB-D panoramas, are used to collect very large-scale indoor scenes by **Armeni et al.** (2017) and **Chang et al.** (2017). To obtain the groundtruth label, each panorama is annotated separately, and then aggregated via voting to annotate the mesh. The labeling process used in ScanNet (**Dai et al.** 2017a) is adopted with crowd sourcing.

Geometry Correction Indoor scene reconstruction with RGB-D scanning has made remarkable advances in recent years. With the state-of-the-art reconstruction algorithms (**Dai et al.** 2017b; **Newcombe et al.** 2011a; **Whelan et al.** 2016), dense large-scale 3D reconstruction achieves very faithful representation of the real world. However, due to the sensor limitation and imperfect capture coverage, the 3D model usually contains missing surfaces. To solve this problem, many geometry completion algorithms have been proposed. **Dzitsiuk et al.** (2017) propose to complete shape by detecting planar surfaces and augmenting surfaces by plane fitting. **J. Huang et al.** (2017) further regulate plane extrapolation by estimating intersection. Beyond planar surfaces and with the focus to reconstruct thin arterial structures, Motif is proposed in **Yin et al.** (2014) to reconstruct incomplete cylinder-alike shape via curve editing. They estimate a set of curves to represent cross-section and object axis, and complete the reconstruction by sweeping curves. A similar idea is proposed by **G. Li et al.** (2010) to reconstruct thin structures by representing surface as arterial snakes. With the advance of deep learning algorithms, attempts have been made to learn 3D reconstruction completion (**Dai et al.** 2017c; **Song et al.** 2017). However, the resolution of these methods remains rather low.

Instance Semantic Segmentation Recently, impressive progress has been made in instance semantic understanding. In particular with the success of deep learning algorithms and convolutional neural networks (CNNs). The state-of-the-art networks (**K. He et al.** 2017; **Redmon and Farhadi** 2017), trained on large-scale 2D image datasets such as ImageNet (**Russakovsky et al.** 2015) and MSCOCO (**T.-Y. Lin et al.** 2014), performs at considerably high accuracy in instance semantic predictions. Learning with 3D data is more challenging due to

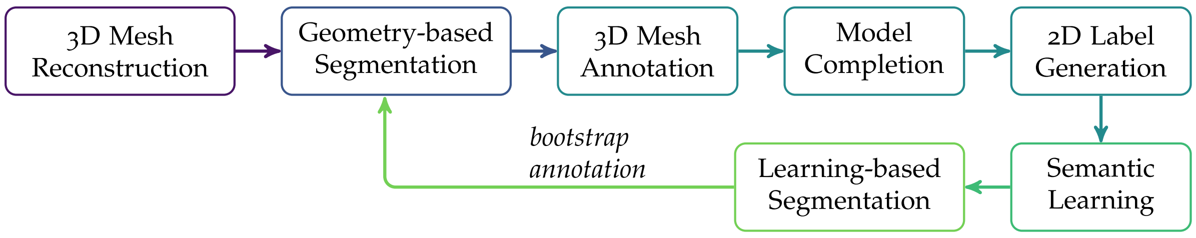


Figure 9.3: The proposed closed-loop pipeline to generate large-scale semantic annotations in 3D. First, the system takes 3D meshes, computes geometry-based segmentations to assist human annotation while allowing free-form labeling. Then, the annotated meshes are rendered to obtain fully annotated 2D sequences, which are used to train deep learning algorithms for semantic understanding. Last, for a new mesh the trained model predicts semantics, fuses the prediction into a geometry-based 3D segmentation and bootstraps annotation.

the fact that 3D data are typically unstructured. Early attempts extend 2D convolution to 3D volumetric data (**Dai et al.** 2017a; **Dai et al.** 2017c; **Song et al.** 2017), where the resolution is highly limited by memory. Recently, attempts have been made to learn directly from point clouds (**Qi et al.** 2017a; **Qi et al.** 2017b) or over the geometric manifold of meshes (**Masci et al.** 2015; **Monti et al.** 2017).

9.3 Algorithm Overview

Figure 9.3 shows the proposed closed-loop pipeline to generate large-scale semantic annotations in 3D. First, the system takes 3D meshes, computes geometry-based segmentations to assist human annotation. An annotator then refines and semantically annotates the pre-segmentation using free-form mesh painting. Then, the annotated meshes are rendered to obtain fully annotated 2D sequences, which are used to train deep learning algorithms for semantic understanding. Given enough human segmented and annotated meshes, we can close the loop by using the trained model to infer semantics in image space and fusing these prediction into a consistent segmentation on the 3D mesh.

9.3.1 Preliminaries

A 3D colored mesh M is a manifold defined by a set of vertices $\mathcal{V} = \{\boldsymbol{v}_i \in \mathbb{R}^3\}$, edges $\mathcal{E} = \{\boldsymbol{e}_{ij}\}$, and polygon primitives $\mathcal{P} = \{p_i\}$. The attributes of a vertex $\boldsymbol{v}_i$ contain unit normal $\boldsymbol{n}_i \in \mathbb{R}^3$, and color $\boldsymbol{c}_i \in \mathbb{R}^3$. Mesh M can be partitioned into disjoint segments M_i and assigned to label ℓ_k. The instance and semantic partition is denoted by M_i^s and M_i^q, respectively. An RGB-D recording is a sequence $\mathcal{S}$ of color image, depth image and the correspondin 6DoF camera pose $\left\{I^t \in \mathbb{R}^3, D^t \in \mathbb{R}, \boldsymbol{\xi}^t \in \mathbb{R}^6\right\}$ with time stamp t. In this work, the 6DoF rigid body motion $\boldsymbol{\xi}^t$ is assumed known for each frame. Given a camera model, the projection of a 3D point $\boldsymbol{v}$ onto the 2D image is denoted by $\boldsymbol{x} = \pi(\boldsymbol{v})$, with $\boldsymbol{x} \in \mathbb{R}^2$

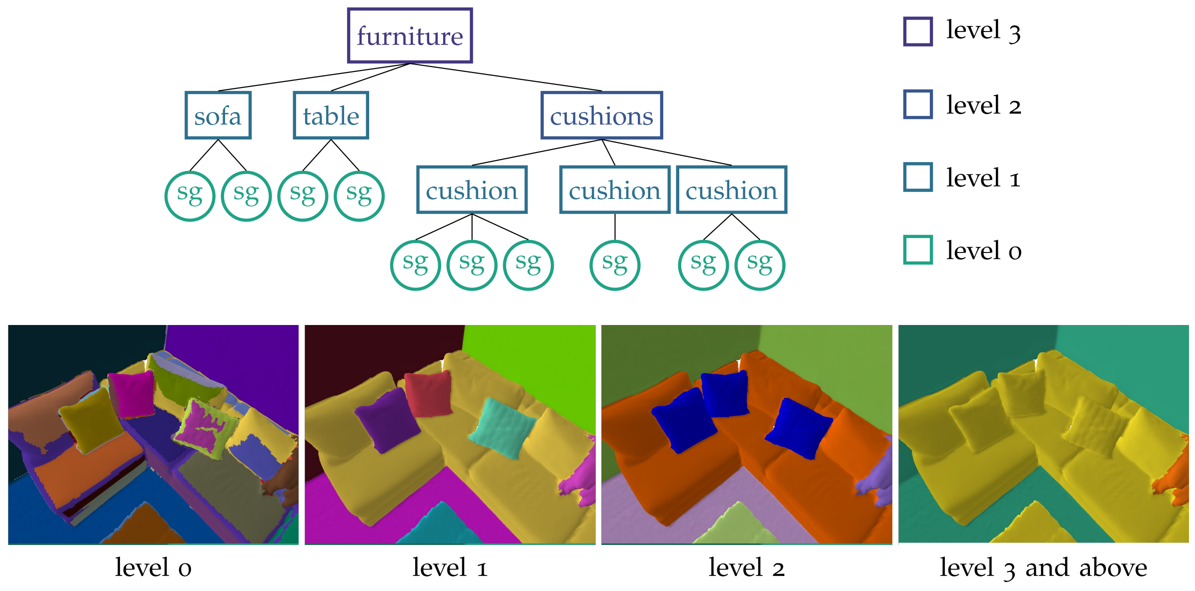


Figure 9.4: A semantic annotation tree encodes different categories and their hierarchical relationship. The level 0 represents only non-semantic segments. The level 1 encodes object instances, where instances of the same category are further grouped into semantic class representations at level 2. The level 3 and the above further encodes the hierarchical relationship between different categories. The second row shows the corresponding label rendering for each tree levels.

and the back-projection by $\mathbf{v} = \pi^{-1}(\mathbf{x})$. To compute color distance, we always use CIELab color space instead of RGB, and $\Delta(\mathbf{c}_i, \mathbf{c}_j)$ denotes the Euclidean color distance in CIELab space.

Annotation is the process of a human interacting with a 3D mesh. To compute where an annotators clicks on the mesh, we use OpenGL to render the polygon index into a frame buffer and read out the index under the mouse position. We find this solution more efficient than ray tracing. Similarly, to obtain 2D labeling from annotated meshes, the label value is looked up through a rendered primitive index.

9.4 Segmentation-Aided Free-form Annotation With Geometry Correction

In this section, we propose an annotation algorithm to generate accurate semantic instance labeling on the 3D mesh and 2D renderings from it. Our goal is to assist humans to efficiently label semantic objects accurately, ensure label consistency between observations from different viewpoints, and maintain a tight association of 3D reconstruction and 2D images.

To this end, the proposed tool operates on 3D colored meshes. The algorithm initializes the annotation with a geometric segmentation. Annotation is an iterative process between joining segments and changing segments to correctly capture details via free-form painting along the mesh manifold. View rendering is applied to propagate the annotation results from 3D to 2D. To compensate rendering errors due to imperfect reconstruction, we introduce human-aided geometry completion.

9.4.1 Hierarchical Annotation with Semantic Tree

The process of annotation is to create a mapping from primitives to the label domain $f : p_i \mapsto \ell_j$. When the mapping f is one-to-one, it is trivial to organize the results. However, considering how annotated models are used in practice, it is desirable to support a multi-valued mapping, *i.e.*, the same primitive can be a rug in for one application, and floor for another. Similarly, hierarchical relationships are fundamental to semantic relationships. For example almost any object can be broken down into smaller parts or grouped into new categories. To support one-to-many mapping and hierarchical relationships in the segmentation, we introduce the semantic tree. Figure 9.4 illustrates this idea. In our work, leaf nodes contain sets of primitives that correspond to segments generated either by the pre-segmentation algorithm or free-form painting. These leaves are connected into a tree to represent different semantics. In this work, the hierarchy is defined as follows: level 0 represents the aforementioned non-semantic leaf segmentations, level 1 represents object instances, level 2 represents object classes, level 3 and above encodes higher-level semantic sets. With this definition, rendering the tree at different levels naturally yields different aspects of the annotation. This is shown in Figure 9.4, up to level 3 of the semantic tree.

9.4.2 Geometry-based Mesh Segmentations

To extract a sensible initial segmentation, we rely on the following robust planar segmentation algorithm: (1) run directional segmentation via the DP-vMF-means algorithm by **Straub et al.** (2014) and (2) run connected component analysis for each of the directional segments along the manifold of the mesh. We classify a segment as planar by analyzing the eigenvalues $\lambda_1 < \lambda_2 < \lambda_3$ of the covariance matrix of all points in the segment. Only if both $\lambda_1 \ll \lambda_2$ and $\lambda_1 \ll \lambda_3$ do we accept the segment as a plane. We refine any segments of the mesh that are not classified as planar via the Felsenszwalb segmentation algorithm (**Felzenszwalb and Huttenlocher** 2004). This leads to a finer segmentation of all non-planar segments that are more useful for the next step of free-form segmentation adjustment and semantic annotation.

To further clean up the resulting segmentation we run a 3D bilateral filter as the final step in the pre-segmentation. The bilateral filter works along the manifold of the mesh with the following weighting function:

$$\omega_j = \exp\left(-\frac{\Delta(\mathbf{c}_j, \mathbf{c}_i)^2}{\sigma_c^2} - \frac{|\mathbf{v}_j - \mathbf{v}_i|^2}{\sigma_v^2}\right) . \tag{9.1}$$

9.4.3 Segmentation Refine with Free-form Painting

In many situations, joining segments does not lead to the desired annotation. This is because 1) boundary of segments often misaligns with the actual object, 2) some segments

ALGORITHM 9.1: Free-form painting along 3D mesh manifold.

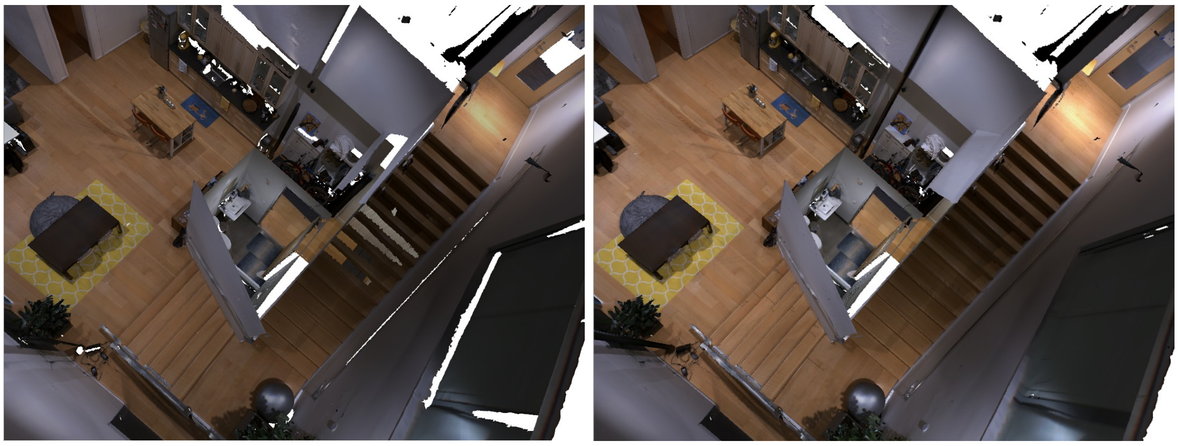
```
1  Input: 3D colored mesh $M$ and a seed primitive $p_s$
2  Parameter: $\alpha_t$ angle threshold, $d_t$ distance threshold, $\delta_t$ color threshold
3  Output: New segment $M_k$
4  queue: $Q \leftarrow p_s$
5  while $Q \neq \varnothing$ do
6      pop primitive $p_k$
7      if $p_k \in M_k$ then
8          continue
9      $N_j$ = findNeighboringPrimitives($p_k$)
10     for each neighbor $p_k$ in $N_j$ do
11         if $|\mathbf{n}_k^T \mathbf{n}_s| > \cos\alpha_t$ and $\|\mathbf{p}_k - \mathbf{p}_s\| < d_t$ and $\Delta(\mathbf{c}_k, \mathbf{c}_s) < \delta_t$ then
12             $Q \leftarrow p_k$; $M_k \leftarrow p_k$
13 return $M_k$
```

connect multiple object parts, and 3) heuristic pre-segmentation algorithms fail to distinguish objects, *e.g.*, to separate rug and floor. To solve this problem, SceneNN (**Nguyen et al.** 2017) generates segmentations, and allows annotators to break large segments by switching from coarse to fine segmentation. We argue that a pre-segmentation should not limit annotations, but rather support the annotator. Therefore, we propose free-form painting along mesh manifold to refine segments. From a user selected seed primitive, Algorithm 9.1 describes the painting method. It uses region growing along the mesh to locate segments. Region growing is regulated with three parameters. Parameter α_t regulates the smoothness by comparing the normal direction, parameter d_t limits the Euclidean distance to the seed primitive, and parameter δ_t regulates the color similarity. With a proper combination of these parameters, the region growing is flexible in selecting any surface patches, from large planar structures to small curved areas. It also enables extracting detailed texture patterns, such as posters on the wall.

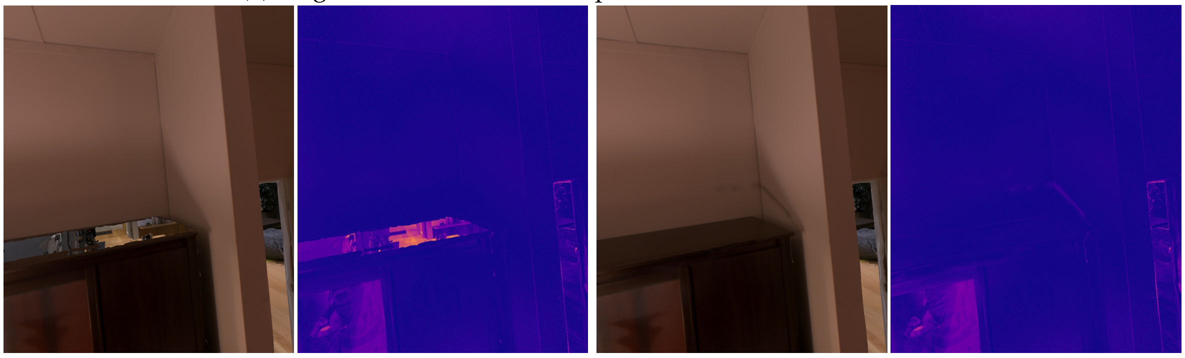
The free-form painting introduces topology changes to the semantic tree. To reflect these changes and to preserve the tree structure, affected tree branches are detected and their empty twin branches created. Leaves affected by painting are split into two. The unselected primitives remain unchanged, the selected ones are extracted and transferred to the twin branch.

9.4.4 Human-Aided Surface Repairing

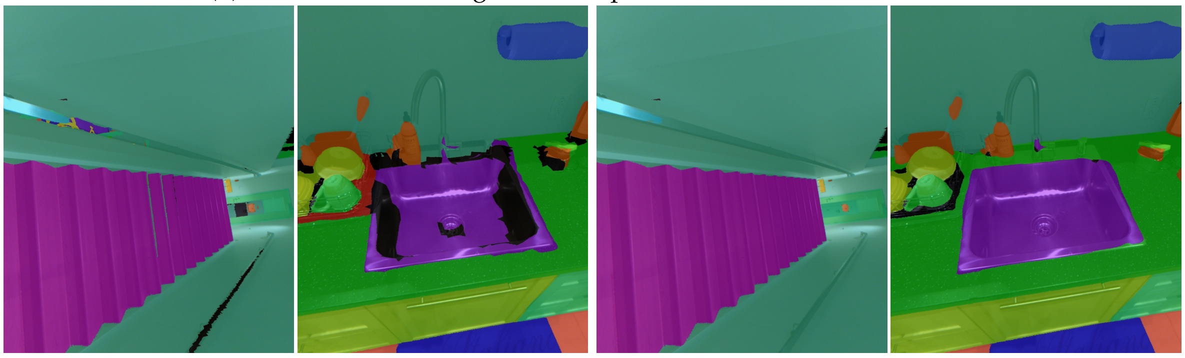
Many indoor reconstructions rely on structure light sensor to obtain dense reconstruction. Due to sensor limitations and imperfect capture coverage, the resultant mesh typically contains missing surfaces. This leads to wrong projection, where surfaces behind the missing reconstructions get projected onto 2D images (see Figure 9.5). Comparing the rendered depth to raw depth images can detect some errors, however, depth is not always available. To maintain a tight association between 3D annotations and 2D images, we propose two



(a) original model $\mathcal{M}$ vs. complete model $\mathcal{M}_c$ with our tool



(b) rendered RGB images and its photometric error of $\mathcal{M}$ vs. $\mathcal{M}_c$



(c) rendered label images by annotated $\mathcal{M}$ vs. annotated $\mathcal{M}_c$

Figure 9.5: The influence of surface repairing on 2D rendering. The top row visualizes a more complete reconstruction produced by our algorithm. The middle row shows the rendering error as the photo-consistency between input RGB and the rendered RGB, with brighter color indicating larger error. The last row compares label rendering, where our method prevents rendering erroneous labels into 2D views and yields more complete annotations.

simple yet effective techniques to repair reconstructions.

Patching of Planar Holes Observing that often missing surfaces are planar, our method lets annotators mark a polygon $\mathcal{B}_i$ around a planar hole. The plane equation parameterized by $\rho_i := (\mathbf{n}_i, d)$ is then estimated by least-square plane fitting,

$$\arg\min_{\mathbf{n}_i, d} \sum_{\boldsymbol{v}_i \in \mathcal{B}_i} \left(\mathbf{n}_i^T \boldsymbol{v}_i + d\right)^2 . \tag{9.2}$$

To stitch the mesh, the planar patches are directly integrated into SDF volume which are used to reconstruction the model. To update the voxels around target holes, the SDF value $F(\boldsymbol{v})$ is calculated as the distance between voxel center $\boldsymbol{v}$ and the target plane

$$\begin{aligned} F(\boldsymbol{v}_i) &= \frac{W(\boldsymbol{v}_i)F(\boldsymbol{v}_i) + \omega(\mathbf{n}^T\boldsymbol{v}_i + d)}{W(\boldsymbol{v}_i) + w} , \\ W(\boldsymbol{v}_i) &= W(\boldsymbol{v}_i) + \omega . \end{aligned} \tag{9.3}$$

Once the SDF values are updated, the mesh is regenerated via the standard Marching Cubes algorithm (**Lorensen and Cline** 1987). The augmented polygons are automatically assigned to the most likely boundary primitive label.

Extrapolation of Cylindrical Structures To complete the missing surfaces of cylindrical structures, *e.g.*, the partially reconstructed pillars in Figure 9.5, we implement a technique inspired by **Yin et al.** (2014). The idea is to first estimate the curve of an objects central axis and cross-section shape, and then to complete the missing surfaces by sweeping the cross-section curve along central axis. Considering our target application we note that there are mostly straight dominant supporting structures that cause rendering errors. Hence, we simplified the original work. Instead of estimating the L1-medial skeleton axis (**H. Huang et al.** 2013) to approximate central axis, we estimate the objects central axis as follows. First the dominant principal component is computed via PCA to approximate the central axis. Then neighboring vertices are projected onto the tangent plane and the cross-section shape is estimated by fitting a non-uniform rational basis spline (NURBS) by **Wang et al.** (2006). Finally the axis is adjust to the center of cross-section NURBS. In addition, the sweeping direction is adjustable by annotators to achieve optimal results.

These two techniques do not optimally correct reconstruction faults. However, they effectively reduce the most common rendering errors and yield better label propagation from 3D model to 2D images.

9.5 Closing Annotation Loop by Learning from Accumulation

Annotation is an expensive process, therefore, it is important to incorporate machine intelligence to assist the human operator. In this section, we describe the close-loop annotation scheme to bootstrap large-scale semantic annotation collection. Inspired by the recent breakthrough of 2D deep learning, this work builds upon the state-of-the-art instance segmentation algorithm Mask-RCNN. To propagate the understanding from multi-view 2D images to obtain consistent predictions of 3D meshes, we develop a novel method for multi-view semantic instance fusion onto a 3D mesh. Furthermore, we introduce a technique to bridge the gap between vision-based and geometry-based segmentation.

9.5.1 Generate Video Annotation

The first step towards closing the loop of annotation is to train machine learning algorithms. In this work, we base our development on Mask-RCNN. In order to train the 2D convolutional neural network, we render the 3D semantic mesh into the corresponding camera view to obtain densely annotated video sequence. Due to the noise and errors in pose estimation, model reconstruction, and camera calibration, the rendered label images do not always fully respect the object boundary in the original color image. The missing labels in mesh annotation and missing surfaces in reconstruction also contribute to noises in label rendering. To correct these artifacts, we apply 2D joint bilateral filter to smooth the rendering, while preserving the edges. Similar to 3D bilateral filter,

$$\omega_j = \gamma_i \exp\left(-\frac{\Delta(\mathbf{c}_j, \mathbf{c}_i)^2}{\sigma_c^2} - \frac{|\mathbf{v}_j - \mathbf{v}_i|^2}{\sigma_v^2} - \frac{|\mathbf{x}_j - \mathbf{x}_i|^2}{\sigma_p^2}\right), \tag{9.4}$$

where σ_c^2, σ_x^2 and σ_v^2 are the variance in color, 2D pixel coordinate and 3D Euclidean. The weight parameter γ_j is an estimation of rendered label certainty. We observe in the experiment that, the rendered label is more prone to error around object boundaries and label jumps. Therefore, a combined edge map is estimated as the strong image gradient plus the label boundary. The weighting map γ is then calculated as the distance towards the nearest edge point and then normalized to the range of $[0, 1]$. The value γ_i is also used to compute a dynamic filter window. Given a maximum window size K, the filter size for pixel i is given by $\gamma_j F$.

9.5.2 Instance Segmentation on a 3D Mesh

The following develops an approach for computing instance-level semantic segmentation in 3D, which more precisely, is to find a tuple of semantic and instance ids, (M_i^s, M_i^q), for each primitive p_i on a mesh M. Shown in Algorithm 9.2, the approach operates in three main phases. First, a Mask-RCNN model is used to detect objects in each image frame I^t in a raw video capture sequence $\mathcal{S}$ (line 5-6). Specially, Mask-RCNN finds a list of object masks $\{\mathbf{d}_k\}$, each of which is associated with a semantic class c_k and a confidence score α_k. Using the projection function found during 3D reconstruction, each of the detections in 2D can be mapped onto the 3D mesh (line 8-10). Each mapped detection consists of the same semantic class c_k, a set of face ids $\mathbf{m}_k$ on the 3D mesh, and the corresponding confidence score $\mathbf{r}_k$ that combines Mask-RCNN's prediction score α_k and the mesh's geometry information (*i.e.,* camera pose with respect to the mesh). In the second phase, the set of individual detections $\mathcal{D}$ are fused together to obtain semantic-level segmentation $\mathbf{s}$ as shown in line 10. In the third phase, since each instance is constrained to have one semantic id, instance-level

ALGORITHM 9.2: 3D instance segmentation.

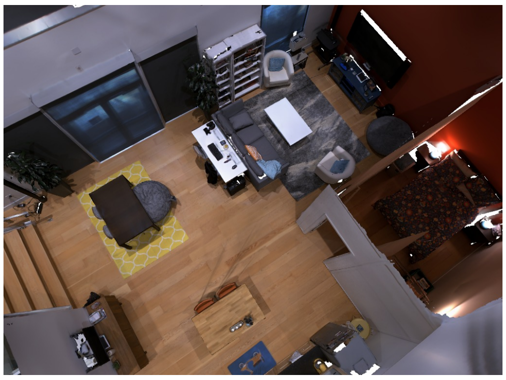
```
1  Input: A video sequence $\mathcal{S}$ and camera model for projection $\pi$
2         A Mask-RCNN model $\mathbf{R} : I \mapsto (c_1, \alpha_1, \mathbf{d}_1), \ldots, (c_{I_n}, \alpha_{I_n}, \mathbf{d}_{I_n})$
3  Output: Semantic and instance segmentation $M^s, M^q$
4  $\mathcal{D} \leftarrow \varnothing$        // initialize set of instance detections on mesh
5  for each image $(I^t, \xi^t)$ in $\mathcal{S}$ do
6      $(c_1^t, \alpha_1^t, \mathbf{d}_1^t), \ldots, (c_{I_n}^t, \alpha_{I_n}^t, \mathbf{d}_{I_n}^t) \leftarrow \mathbf{R}(I^t)$
7      for each detection $\mathbf{o}_k^t = (c_k^t, \alpha_k^t, \mathbf{d}_k^t)$ do
8          $(c_k^t, \mathbf{r}_k^t, \mathbf{m}_k^t) \leftarrow$ projectionAndAggregate $(\mathbf{o}_k, I^t, \xi^t; \pi)$
9      $\mathcal{D} \leftarrow \mathcal{D} \cup \left\{(c_1^t, \mathbf{r}_1^t, \mathbf{m}_1^t), \ldots, (c_{I_n}^t, \mathbf{r}_{I_n}^t, \mathbf{m}_{I_n}^t)\right\}$
10 $M^s \leftarrow$ semanticFusion($\mathcal{D}$)
11 $M^q \leftarrow \mathbf{0}$
12 for each semantic unique class $s_k$ in $M^s$ do
13     $\mathcal{D}_{s_k} \leftarrow \{\mathbf{o} | \mathbf{o} \in \mathcal{D}$ s.t. detection $\mathbf{o}$ has classId $s_k\}$
14     $\mathbf{m}_{s_k} \leftarrow \{m_i | s_{m_i} = s_k\}$
15     $M_{s_k}^q \leftarrow$ instanceFusion($\mathcal{D}_{s_k}$, $\mathbf{m}_{s_k}$)
16     $M^q \leftarrow$ extendIds($M_{s_k}^q$)
17 return $\mathbf{s}, \mathbf{q}$
```

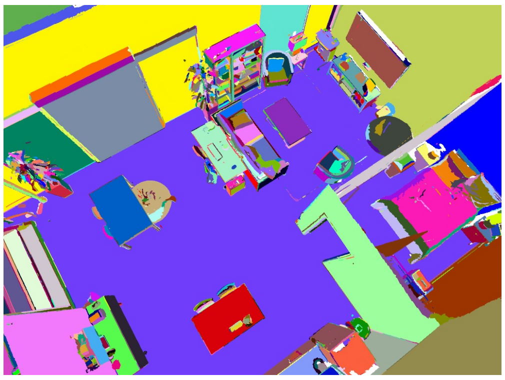
segmentation can be performed separately for each semantic class s_k, as shown in lines 12-16. Details of the latter two phases are presented below.

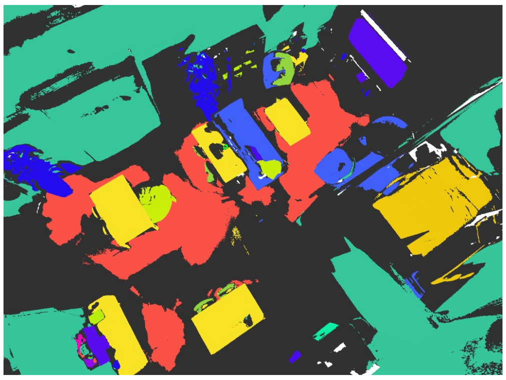
Semantic Fusion Outlined in Algorithm 9.3, semantic fusion finds a partition $\mathbf{s}$ of the mesh by aggregating individual detections in $\mathcal{D}$. Specifically, the confidence counts $\mathbf{r}_k^t$ of each detection are cumulated over the instance's 3D volume $\mathbf{m}_k^t$ (set of face ids on the mesh) in lines 6-8; and a partition is obtained by assigning each face id to the semantic class with the most counts in line 9. The aggregation process is repeated several times to remove detection instances that are inconsistent (line 7) with the current partition, which is important for removing false-positive detections by Mask-RCNN. In this work, consistency is determined by whether more than half of the detection volume $\mathbf{m}_k^t$ is assigned the predicted semantic class c_k^t in the partition M^s.

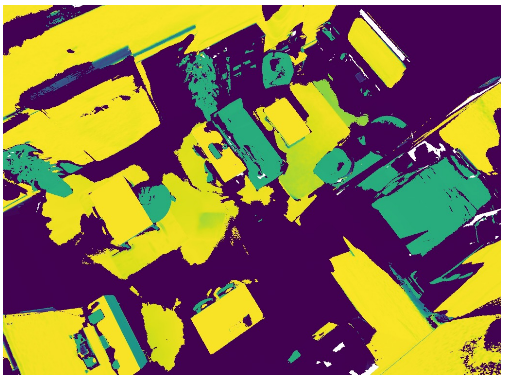
Instance Fusion Instance-level segmentation is more difficult because instance ids between different observations cannot be associated directly. For example, an chair in image A could correspond to any or none of the chairs in image B. More importantly, since the detections are originally derived from 2D images, different detections of the same object instance could correspond to images from different view points and thus have little in overlap on the 3D mesh. To address this challenge, Algorithm 9.4 is developed to find an instance-level partition $M_{s_k}^q$ when given a set of detections $\mathcal{D}_{s_k}$ for each semantic class s_k.

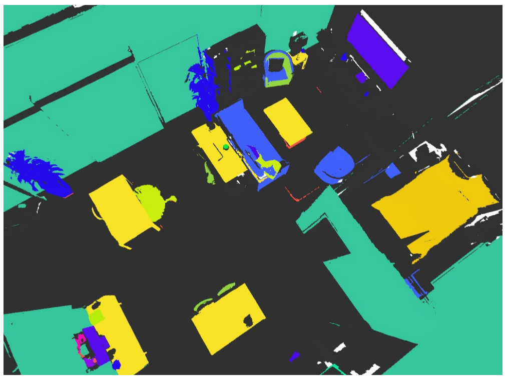
The proposed method works similarly to expectation maximization. In particular, the algorithm iterates between (i) finding a soft assignment $\mathbf{z}^t$ for each detection (line 9-14), and (ii) aggregating the assigned detections' confidence counts $\{\mathbf{u}_h\}$ to update the instance partition

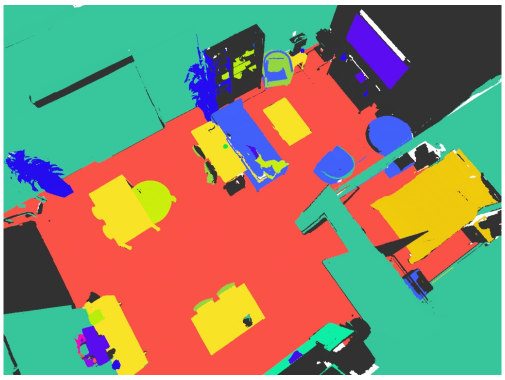

(a) target reconstruction

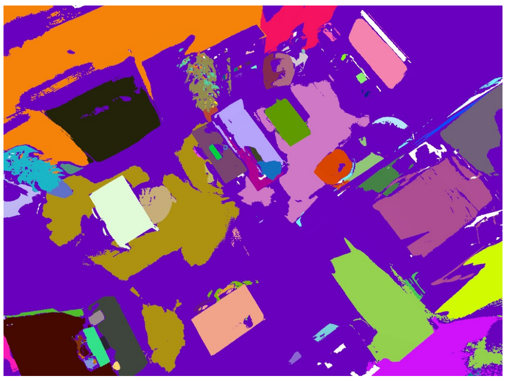

(b) geometry-based segmentation


(c) Mask-RCNN semantic prediction


(d) estimated label confidence


(e) uniform semantic transfer


(f) weighted semantic transfer


(g) Mask-RCNN instance prediction

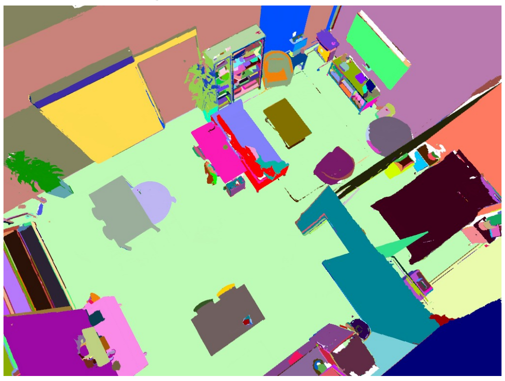

(h) weighted instance transder

Figure 9.6: Conditioned label transfer to integrate Mask-RCNN prediction and geometry-based segmentation.

ALGORITHM 9.3: Semantic fusion

```
1  Input: Semantic class, masks and confidence scores of instance-level detections on a 3D mesh
2      $\mathcal{D} = \left\{(c_1^t, r_1^t, m_1^t), \dots, (c_{I_n}^t, r_{I_n}^t, m_{I_n}^t)\right\}$
3  Output: Semantic partition of the mesh $M^s$
4  $M^s \leftarrow 0$          // initialize a partition vector
5  while not converged do
6      $\{u_h \leftarrow 0\}$ // initialize confidence counts for each semantic class $h$
7      for each detection ($o_k^t = (c_k^t, r_k^t, m_k^t)$) in $\mathcal{D}$ do
8          if first iteration or isConsistent($o_k^t$, s) then
9              $u_{c_k^t} \leftarrow$ cumulateCounts($u_{c_k^t}, r_k^t, m_k^t$)
10     $M^s \leftarrow \arg\max(u_1, \dots, u_s)$
11 return s
```

$M^q_{s_k}$ (line 15-19). In this work, we compute the intersection over union (IoU) of a detection with each of the existing partitions (line 11-13). [1] A key distinction here from the classical clustering problem is that an observation t can have multiple correlated detections. [2] For example, two detections from the same image should correspond to separate instances, and thus the soft assignment step in line 14 needs to take into account the IoU between each pair of detections and instance partitions. Furthermore, for semantic classes with instances of large physical size, such as door and wall, each detection typically only gets a small, partial view of an object instance. The IoU matrix can also be used as a metric to determine whether multiple partitions should be merged into a single object (line 17-18).

9.5.3 Conditioned Label Transfer

At the early stage of data collection, available annotated data is rather limited. This leads to the fused semantic and instance segmentation from Mask-RCNN fails to always make correct predictions for every mesh primitive. Shown in Figure 9.6(c) as an example, it can be seen the half of the mesh do not get predictions. However, given our conservative fusion strategy, most valid predictions are correct. Notice that the geometry-based segmentation shown in Figure 9.6(b) also provides clean and useful predictions on object classes. In particular, the structural surfaces, *e.g.*, floors and walls, are well-segmented. Based on this observation, we proposed to transfer the semantic predictions conditioned on the geometry segments.

For this purpose, one trivial solution is to use maximum likelihood (ML) estimation, by assigning each segment the label that dominants the corresponding segment. The ML estimation does not yield optimal solution, as shown in Figure 9.6(c). The correct labels are rejected due to a slightly lower occurrence frequency. To resolve this problem, we propose

[1] To account for a camera's limited field of view, the union calculation between a detection and an instance partition is restricted to the part of the mesh visible to a detection's corresponding image.
[2] Recall Mark-RCNN can find multiple instances of the same semantic class in an image.

ALGORITHM 9.4: Instance fusion

```
Input: Instance-level detections from a single semantic class,
        $\mathcal{D}_{s_k} = \left\{(c_1^t, \mathbf{r}_1^t, \mathbf{m}_1^t), \ldots, (c_{I_n}^t, \mathbf{r}_{I_n}^t, \mathbf{m}_{I_n}^t)\right\}$ s.t. $c_h^t = s_k \, \forall t, h$
        A set of mesh indices $\mathbf{m}_{s_k}$ whose semantic class is $s_k$
Output: Instance-level partition $M_{s_k}^q$ for the set of indices $\mathbf{m}_{s_k}$
$M_{s_k}^q \leftarrow$ randomPartition($\mathbf{m}_{s_k}$)        // initialize instance partition
while not converged do
    $\{\mathbf{u}_h \leftarrow 0\}$ // initialize confidence counts for each instanceId $h$ in $\mathbf{q}$
    $\mathbf{S} \leftarrow \mathbf{0}$    // initialize an instance merge/split count matrix
    for each observation $t$ in $\mathcal{D}_{s_k}$ do
        $\mathtt{iouMat} \leftarrow \mathbf{0}$
        for each of kth detection $(c_k^t, \mathbf{r}_k^t, \mathbf{m}_k^t)$ from observation $t$ do
            for each instanceId $h$ in $M_{s_k}^q$ do
                $\mathtt{iouMat}[k, h] \leftarrow$ computeIoU($\mathbf{r}_k^t, \mathbf{m}_k^t$, $M_{s_k}^q$, $h$)
        $z^t \leftarrow$ assignDectectionToInstance($\mathtt{iouMat}$)
        for each of kth detection $(c_k^t, \mathbf{r}_k^t, \mathbf{m}_k^t)$ from observation $t$ do
            $\mathbf{u}_{z_k^t} \leftarrow$ cumulateCounts($\mathbf{u}_{z_k^t}, \mathbf{r}_k^t, \mathbf{m}_k^t$)
        $\mathbf{S} \leftarrow$ updateMergeSplitCounts($\mathbf{S}, \mathtt{iouMat}$)
    $\{\mathbf{u}_h\} \leftarrow$ mergeSplitInstance($\mathbf{S}, \{\mathbf{u}_h\}$)
    $M_{s_k}^q \leftarrow \arg\max(\{\mathbf{u}_h\})$
return q
```

to estimate the label confidence based on the semantic meaning and the assumption of Manhattan world. Taking a Mask-RCNN segmentation, the gravity direction is first estimated by taking the dominant segments that are labeled floor. The least square plane fitting is then performed to obtain the parameter $(\mathbf{n}_g, d_g)^T$. The estimated plane normal serves as gravity direction. For primitive p_i that is predicted to be floor, the label confidence is estimated by

$$\eta_i = \mathbf{n}_i^T \mathbf{n}_g \cdot \exp\left(-(\mathbf{n}_g^T \mathbf{v}_i + d_g)^2\right) \ . \tag{9.5}$$

Assuming the reconstruction is a Manhattan world, objects that are typically perpendicular to gravity direction, *e.g.*, desk and ceiling, are assigned confidence $\eta_i = |\mathbf{n}_i^T \mathbf{n}_g|$. Whereas, objects that are typically parallel to gravity, *e.g.*, wall, doors and monitors, are assigned confidence $\eta_i = 1.0 - |\mathbf{n}_i^T \mathbf{n}_g|$. For object classes where the surface orientation is difficult to be described in by Manhattan world, the confidence is set to a empirical values which reflect the overall prediction confidence. Such confidence map is showed in Figure 9.6(d). Using the confidence values, a weighted voting is performed in label transfer, which leads to obvious improvements in Figure 9.6(f).

9.6 Experimental Results

9.6.1 Groundtruth Generation

To examine the efficiency and accuracy of the 3D annotation algorithm and the consistency from propagating 3D labels to 2D image, we capture one apartment in great details, and apply the geometry correct techniques described in Section 9.4.4 to fill-in missing surfaces. The comparison of the original and the corrected reconstruction is visualized in Figure 9.5. This model is carefully annotated by a professional annotator to obtain a highly detailed label for every primitive and the result is used as ground truth for further evaluations. Figure 9.7 shows the annotation for both instance and semantics. This golden standard annotation is referred to GTApt in the remaining section.

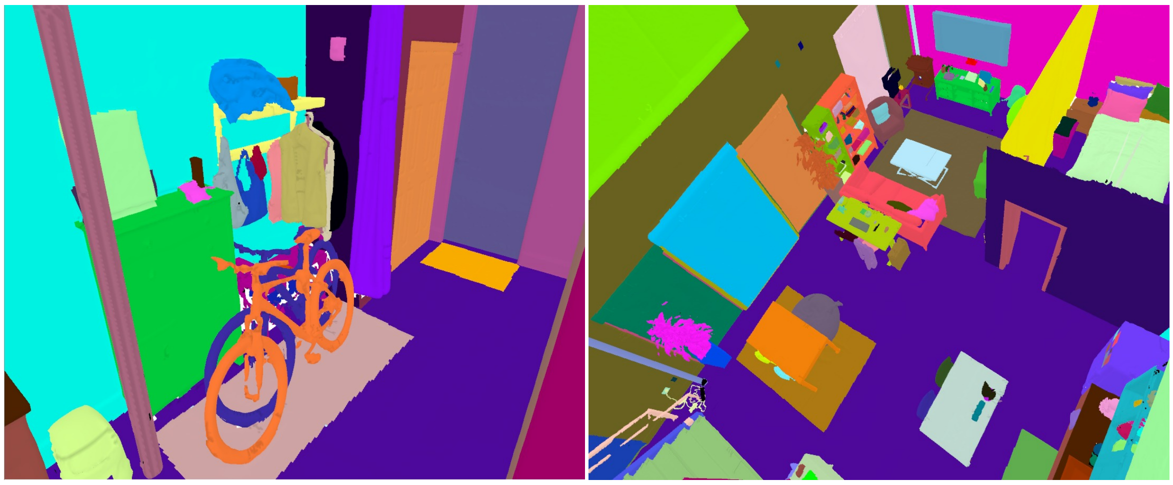
To evaluate the proposed close-loop annotation pipeline, we further collect 20 scans of offices and process them with our tool, naming BS20Scene (Figure 9.8). Using the techniques described in Section 9.5.1, we rendered 8475 labeled images with precise instance-level semantic segmentation, consisting 32 common categories, such as chairs, dining tables, and floor. To achieve better generalization and prevent over-fitting, we add MSCOCO into training Mask-RCNN. To combine with our BS20scene, a subset of MSCOCO with 50 frequent indoor categories were added to training, which amounts to 68 classes of 102147 images.

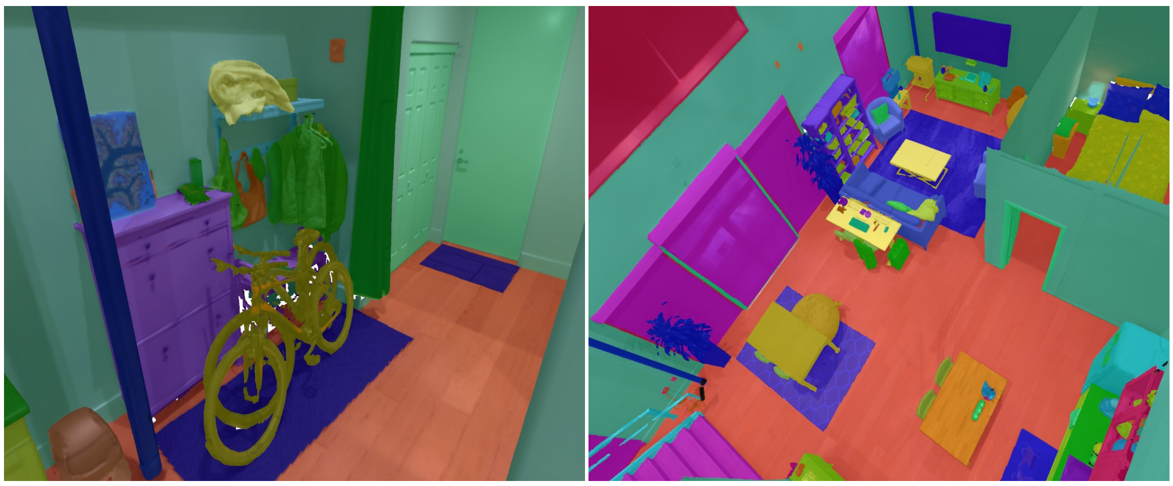
9.6.2 Evaluations

In first experiment, we evaluate the annotation efficiency and accuracy of the proposed segment-based free-form annotation method. In order to obtain fair comparison to the state-of-the-art segment-based annotation methods by **Dai et al.** (2017a) and **Nguyen et al.** (2017), we compute mesh segmentations with different algorithms. Taking the human annotation GTApt as ground truth, and the theoretical optimal accuracy of segment-based annotation is calculated. Assuming annotators only combining segments to annotate and they make no mistake, the optimal accuracy is calculated by

$$A^* = \frac{1}{N} \sum_{i=1}^{N} \frac{|M_i \cap M_k^g|}{|M_i|} , \tag{9.6}$$

where the i-th segment M_i is associated to the k-th ground-truth segment M_k^g by the highest IoU score. Taking three segmentation results from BS20Scene, the optimal parameter is searched for each algorithm and the results are reported in Table 9.1 and the visual segmentation comparison is shown in Figure 9.9. The baseline is Felzenszwalb segmentation (FS), for which we also report the accuracy with its extreme parameter settings. It can be observe that the theoretical optimal accuracy typically increase with a higher number of segments. Ideally, when each primitive is a segment, the accuracy will reach 100%. However, larger amounts of segments indicates higher demand for human annotation. The proposed PlaneExtractor usually yields smaller amount of segments. Overall, combining PlaneExtractor with Felzenszwalb and smooth the results with 3D bilateral filter gives best performance. The result from Table 9.1 also shows that segment-based annotation is prone to 5% error, which is the accuracy gain obtained by amending segments as proposed by our method.


(a) instance mesh annotation


(b) semantic mesh annotation (overlaid with the colored mesh)

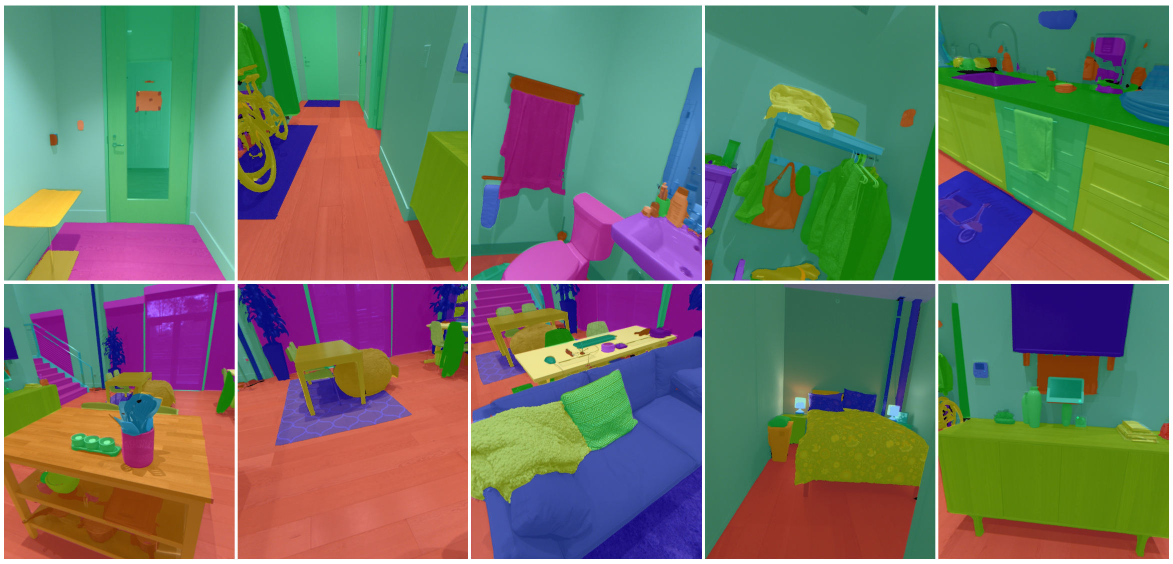
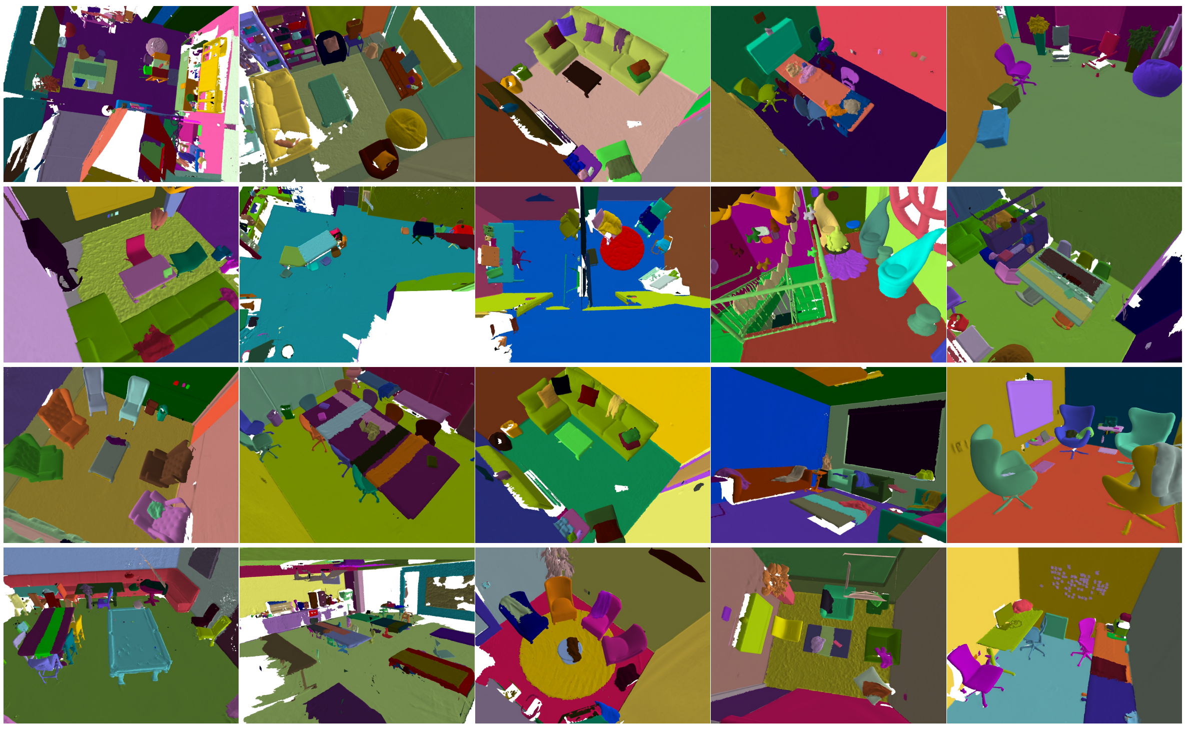

(c) semantic 2D label rendered into camera view (overlaid with the RGB images)

Figure 9.7: The ground-truth apartment annotation created by a professional 3D annotator using our tool, which contains 300+ object instances from 60+ semantic categories.


(a) BS20Scene instance annotation.

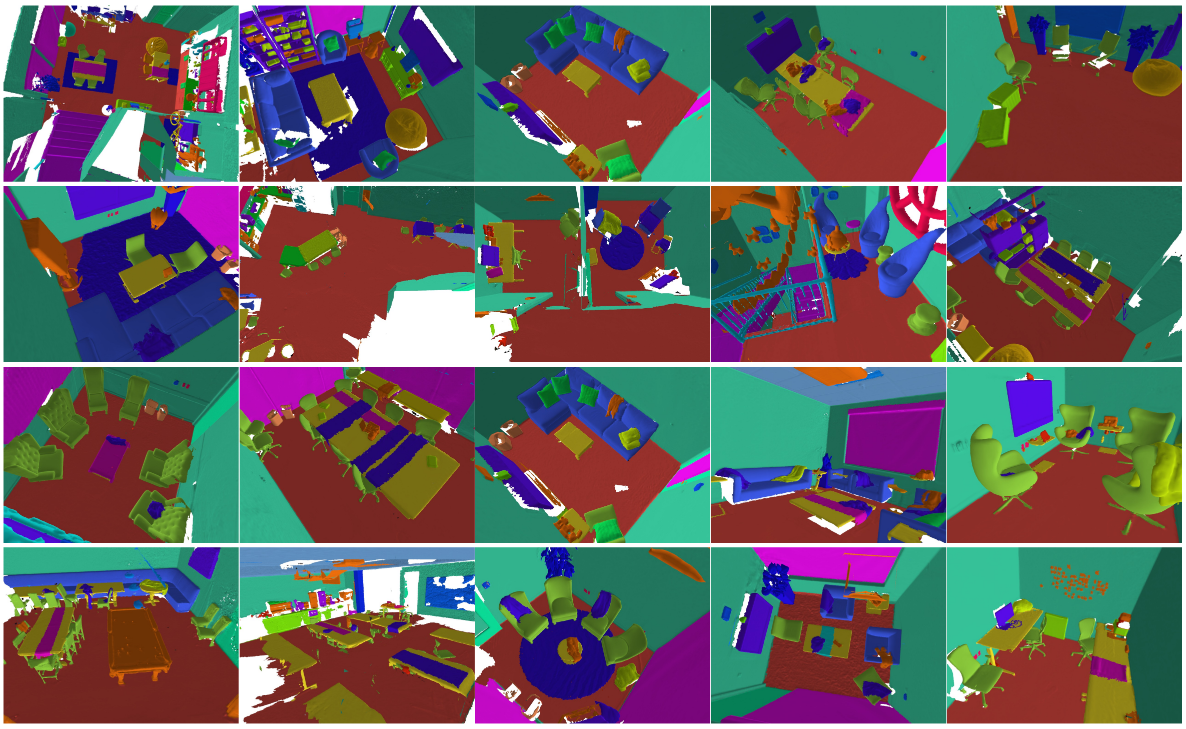

(b) BS20Scene semantic annotation.

Figure 9.8: The BS20Scene dataset annotated by our method and used to generate 2D labels to bootstrap automatic annotations of large scale office labelling as shown in Figure 9.1 (showing the views without structure completion).

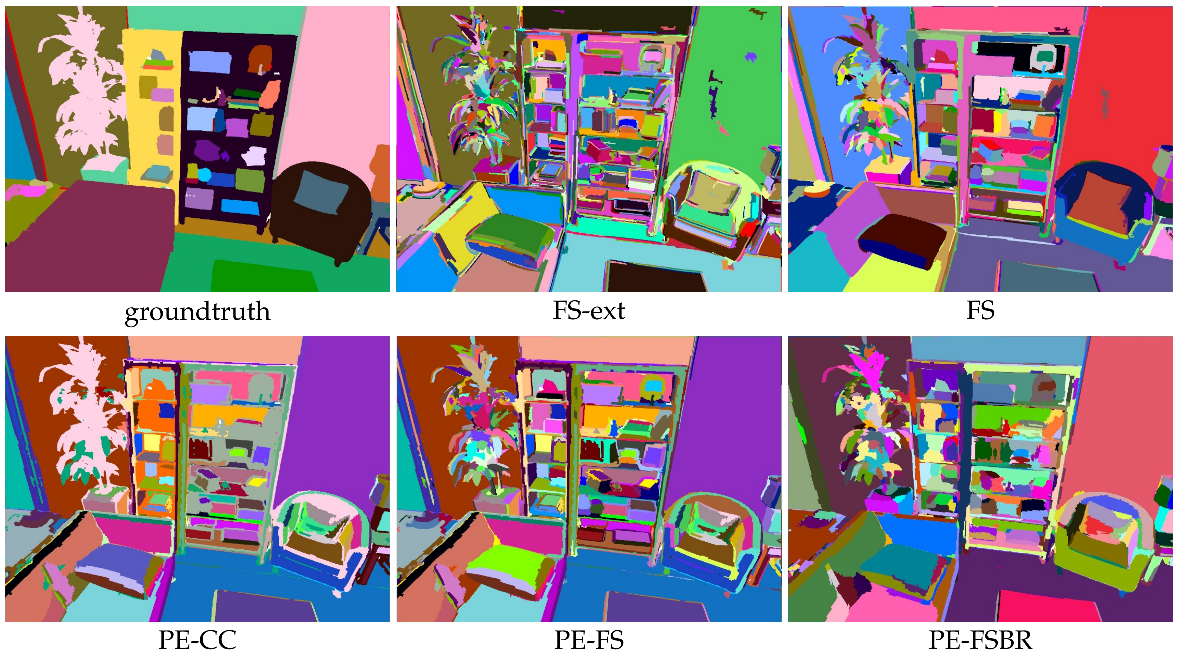


Figure 9.9: Comparison of mesh segmentation algorithms. The theoretical optimal accuracy for each algorithm is presented in Table 9.1.

Table 9.1: Theoretical optimal accuracy for segment-based mesh annotation, comparison between segmentations produced by Felzenszwalb(FS), its extreme setting (FS-ext), PlaneExtractor with connected components (PE-CC), PlaneExtractor with Felzenszwalb (PE-FS) and additionally with 3D bilateral refinement (PE-FSBR).

	room A		room B		room C	
algorithm	num. seg	optimal acc	num. seg	optimal acc	num. seg	optimal acc
FS-ext	21,137	97.38%	3,053	96.90%	2,124	98.02%
FS	4,446	94.48%	620	95.31%	391	95.70%
PE-CC	1,318	91.14%	1,199	94.00%	642	92.30%
PE-FS	2,830	94.03%	1,376	95.54%	786	94.09%
PE-FSBR	2,425	93.65%	527	94.53%	414	93.17%

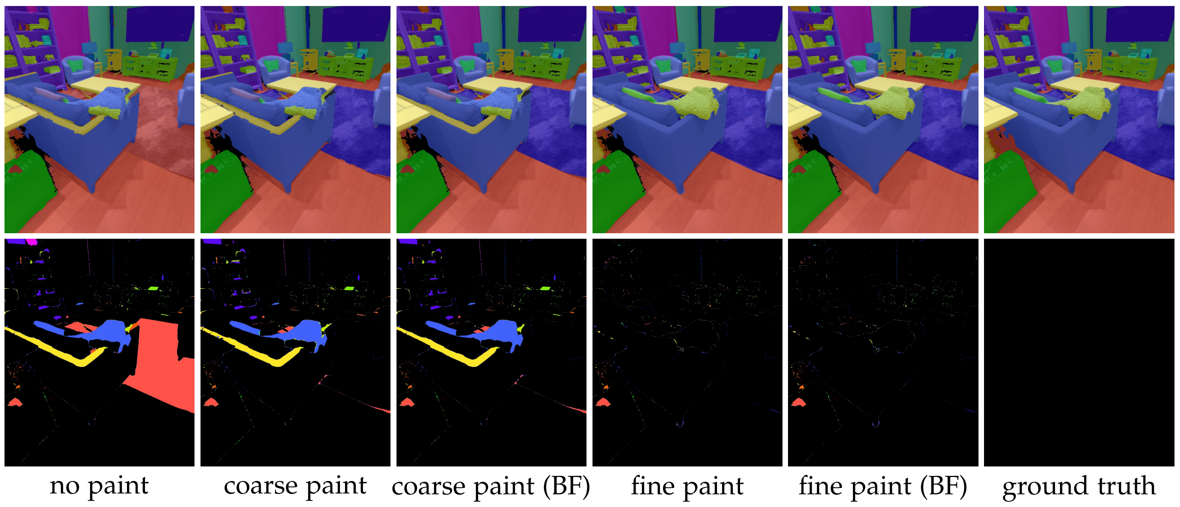


Figure 9.10: Rendered 2D label images with different annotation in comparison to ground truth (BF indicates with bilateral filtering). The corresponding quantitative results are given in Table 9.2.

Many practical applications require 2D annotated images. In the second experiment, we investigate how different annotation influence the consistency on label propagation from 3D meshes to 2D images. We also investigate the effectiveness of the proposed surface correction on rendering. For comparison, the ground-truth labels of the GTApt reconstruction (contain surface correction) is rendered into the original RGB camera trajectory that is used to reconstruct the model. Taking every 10th frames, we obtain approximately 4000 views for quantitative evaluation. Three annotations are obtained as follows: a) annotation by joining segments, b) annotation by adding 30min free-form painting to amend results from a), and last a fully detailed annotation from our method. Table 9.2 gives the quantitative results and Figure 9.10 shows visual comparison. The results shows that segment-based annotation yields the lowest accuracy, whereas a quick adjustment of segments increases accuracy by 10%. The major reason is explained in Figure 9.10, where the pre-segmentation algorithm fail to extract rug from the floor. We argue, these scenarios are very common in real data, which motivates the importance of the ability to correct the segmentation. Comparing the rendering before and after 2D bilateral filter, it can be seen label boundary become better aligned to objects. Figure 9.10 also shows the effectiveness of surface correction, which prevents rendering the floor into computer through the missing surface. This rendering improvement is further evaluate by comparing the photo-consistency between rendered colored image and input RGB. As shown in Table 9.3, with model completion the photometric error decreases and the amount of pixels that are closer to the input RGB images increases.

Table 9.2: Accuracy of 2D rendered semantic label in comparison to geometry corrected detailed annotations.

join segment	coarse paint	fine paint	2D refine	render accuracy
✓				76.71%
✓			✓	77.01%
	✓			87.22%
	✓		✓	87.66%
	✓	✓		96.93%
	✓	✓	✓	97.36%

Table 9.3: Photometric consistency between input and rendered RGB images from reconstruction. The per pixel error is computed in CIELab color space, and Δ-10 is the percentage of pixels within 10 color distance. The results * denote the subset of images that are affected by geometry correction.

algorithm	average	std	Δ-10
baseline	8.47	12.93	78.9%
with completion	8.25	11.86	79.1%
baseline*	8.27	12.70	79.2%
with completion*	7.79	10.70	79.4%

9.7 Conclusions

This work presented an annotation tool for producing high-quality 3D reconstruction and semantic segmentation datasets. In particular, existing 3D datasets often have large areas with missing or incorrect geometry information (*e.g.*, thin structures and holes in dimly lit areas) due to limitations in sensors technologies and data capture procedures. The proposed method addressed this challenge by allowing human annotators to amend the missing geometries. Furthermore, the work provided an efficient method for annotating object instances and semantic information directly on the reconstructed 3D mesh. The annotated 3D data was used to render 2D images for training a state-of-the-art object detection model. Finally, a novel semantic fusion algorithm is developed, which combines object detections from individual 2D images onto the 3D mesh. This closes a 2D/3D loop, where the ground truth data is used to train/improve an object detection model, which subsequently, generates initial annotation (for human annotators to refine) on new data sequences. The proposed approach is evaluated extensively on an indoor dataset of 20 rooms, and used to generate automatic semantic annotation for a large office space.

PART III

Supplementary Materials

All models are wrong, but some are useful.

– George E. P. Box

CHAPTER 10

Dense RGB-D Reconstruction

DENSE 3D colored models of real-world environments are of high relevance to many applications including mixtured/virtual reality, serious gaming, professional civil engineering and 3D printing. In this chapter, we discuss algorithms to obtain such detailed models from common RGB-D SLAM methods. Two scenarios are considered. First, we consider an off-line processing, where the input consist of a triangular mesh and a set of multi-view RGB-D images with known 6 DoF. The primary objective is to produce visually appealing 3D colored models that are robust to various noise in data acquisition. In the second setting, we consider an on-line volumetric color fusion for KinectFusion **Newcombe et al.** (2011a) alike RGB-D SLAM algorithms. The objective in this scenario is to obtain a detailed reconstruction given the trade-off between tracking accuracy and limited GPU memory.

10.1 Dense Colored Model from Multi-view Textures and Triangular Mesh [1]

In this section, we consider an off-line processing to obtain dense colored mapping from an input consists of a triangular mesh and a set of RGB-D images with given poses. An example input is shown in Figure 10.1.

10.1.1 Related Works

Given a mesh model reconstructed from RGB-D SLAM and the multi-view textures, the object is to obtain a consistent colored model. To this end, we need to be aware of the various noise from data acquisition. First of all, textures from commodity RGB-D cameras have relatively low quality, *e.g.,* low pixel resolution, image blur caused by motion blur and distortions due to rolling shutter cameras. Second, brightness is not consistent, due to

[1] © 2013 IEEE. Textual materials, figures, tables reused with permission from Lingni Ma, Luat Do, Egor Bondarevand Peter H. N. de With, *3D Colored Model Generation Based on Multiview Textures and Triangular Mesh.* in proc. of Seventh International Conference on Distributed Smart Cameras (ICDSC), Oct 2013.

(a) mesh with the viewpoint of the camera (b) samples of multi-view textures

Figure 10.1: Example input to the proposed algorithm, consisting of a triangular mesh and a set of multi-view textures with known poses.

uncontrolled lighting during data capturing, auto exposure settings, and varying brightness from non-Lambertian materials. The third error source comes from the imperfect camera calibration and errors from motion estimation. Additionally, mesh reconstruction does not accurately reflect the true geometry, with over smooth geometry, missing reconstructions and wrong surface modeling. With these error sources in mind, it is desirable to have robust coloring algorithms.

In computer graphics, texture mapping is one of the most applied techniques to generate colored models, where each mesh polygon is associated with a 2D texture patch. Textured models provide an efficient representation to embed visual information, In comparison to store color by mesh vertices, textured model can store more detailed appearance context. These advantages in particular benefit meshes with large-sized polygons. The drawback of textured models is the inflexible structure. With the separation between geometry and appearance, the surface continuity is not often not maintained in the texture domain, which creates difficulties in applying image filtering techniques to the disconnected texture patches. To combine the flexibility from colored mesh and the representation capacity from texture, **Yuksel et al.** (2010) proposed the *mesh colors* representation. With mesh colors, color is not only assigned to vertices, but color samples are also interpolated for edges and faces such that their appearance can be better presented even with large triangles.

When using multi-view textures to color a mesh, a key concern is how to obtain proper coloring given the inconsistent observations from multiple viewpoints. One solution is to apply texture blending. To this end, **Lensch et al.** (2003) proposed a triangle-based blending algorithm to smoothen adjacent triangles that are mapped to different viewpoints. The blending performance is highly sensitive to the triangle size, where small triangles are under filtered and large triangles are overly smoothed. To address this problem, pixelwise blending is developed in (**Alshawabkeh and Haala** 2005; **Grammatikopoulos et al.** 2007). This method synthesizes textures, where pixels from different viewpoints that correspond to the same point in geometry are blended. The quality of the colored model is highly dependent on the resolution of the textures and their selected viewpoints.

10.1.2 The Methods

To develop a flexible representation for robust coloring, our solution is to use 3D surface points as processing primitive to store and filter visual information. To this end, we exploit the structured mesh colors (**Yuksel et al.** 2010) to perform adaptive point upsampling and to produce render-friendly dense maps. We further develop an algorithm to generate multi-view textures for straight-forward texture mapping.

10.1.2.1 Adaptive Point Upsampling and Point-Based Color Blending

We now present a point-based algorithm, which uses 3D surface points as primitive to process and associate color information. These points come from the original vertices as well as from upsampled surface points based on the mesh. The goal is to assign a reasonable color value for each point and output a colored model represented as a colored point cloud constrained in a mesh structure.

In this section, we present our point-based algorithm to generate 3D colored models. We store color information by surface points and provide appropriate coloring via blending. Our algorithm is robust to errors from texture deficiencies, camera calibration and mesh modeling. Let us now describe the three processing steps in more detail.

Associate Mesh Polygons to Multi-view Textures Given a mesh and texture, determine the visible triangles in the given view is the first step prior to any processing. One solution is to use ray tracing to obtain the visible triangle. An more efficient alternative however, is through mesh rendering. The mesh rendering is a process of radiosterilization and depth test. In particular using the functionality from OpenGL, the index of the visible mesh triangles can be directly read out from a frame buffer.

Adaptive Point Upsampling With the attempt to preserve all visual information of a dense RGB video into a point-based representation, the amount of input vertices from a mesh is often insufficient. In order to avoid loss of details, we propose to upsample points before coloring. Naturally, the projection area of a triangle onto a 2D texture directly influences the amount of information needs to be represented by the face. The larger the projection, the more pictorial details are contained. It is therefore reasonable to upsample points propor-

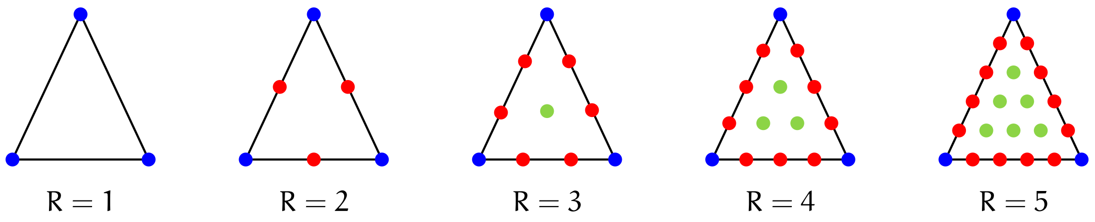


Figure 10.2: Mesh colors with different resolution R, where blue, red and green denote vertices, edge points and face points, respectively.

tional to the projection area. Usually, the projected area varies with respect to the intrinsic triangle size and the camera pose. With these observations, an adaptive point upsampling organized by the mesh colors structure is suitable for our purpose.

As proposed in **Yuksel et al.** (2010), the mesh colors structure is an extension of vertex colors, where colors are not only assigned to vertices, but additional color samples are also added to edges and faces. The mesh colors thus resembles a texture patch, except that the color values are directly associated to the geometry. For the triangular mesh, a resolution factor R is specified for each triangle such that $R-1$ points are added to each edge and $(R-1)(R-2)/2$ points are added to the face. Figure 10.2 depicts some examples of mesh colors structures with different resolution factors. All points belonging to a triangle are evenly spaced, so that they are easily computed using barycentric coordinates with respect to the triangle vertices. Let point $\mathbf{p}_{ij}$ belong to a triangle with $0 \leqslant i \leqslant R$ and $0 \leqslant j \leqslant i$, its barycentric coordinates are given by $\Lambda_{ij} = (i/R, j/R, (1-i-j)/R)$. All points on the triangle can then be calculated by a coordinate matrix Λ, with

$$\Lambda = \begin{bmatrix} \Lambda_{00} & \Lambda_{01} & \cdots & \Lambda_{0(R-1)} & \Lambda_{0R} \\ \Lambda_{10} & \Lambda_{11} & \cdots & \Lambda_{1(R-1)} & \mathcal{O} \\ \vdots & \vdots & & \mathcal{O} & \mathcal{O} \\ \Lambda_{(R-1)0} & \Lambda_{(R-1)1} & \cdots & \mathcal{O} & \mathcal{O} \\ \Lambda_{R0} & \mathcal{O} & \cdots & \mathcal{O} & \mathcal{O} \end{bmatrix}. \tag{10.1}$$

In accordance to Figure 10.2, we denote the coordinates of vertices, edge points and face points with blue, red and green, respectively. The matrix Λ is an upper triangular matrix, where $\mathcal{O} = (0, 0, 0)$ are coordinates of non-existing points.

For adaptive upsampling, we determine the resolution factor R for each triangle according to its maximum projected area α_m among all visible textures. The number of pixels inside the projected triangle serves as an estimation for the projected area. The resolution R is then specified by $R = \lceil \sqrt{\alpha_m} \rceil$ with $R_{min} = 1$. This policy ensures that any three closest color samples that form an equilateral triangle, encompass at most one pixel. Consequently, we approximate the least amount of points required to store the necessary appearance. In this design, upsampled points and original mesh vertices are stored as one point cloud and

(a) color by the closest view criteria

(b) color by smallest view angle criteria

(c) color by proposed blending

(d) the proposed blending with upsampling

Figure 10.3: 3D colored point clouds generated with different color assignment criteria.

referenced to triangles by indices. They can also be arranged in the required format to directly render the mesh colors structure.

10.1.2.2 Point-based Color Blending

Given RGB-D sequences, texture blending reduces undesirable color changes. The unappealing coloring is illustrated in Figure 10.3 (a) and (b), where color values are selected by the closest view distance and smallest view angle, respectively. In addition to abrupt color changes, the errors from imperfect camera calibration as well as inaccurate mesh modeling lead to wrongly assigned color values. To achieve better coloring, the point-based blending is proposed. First, color samples are collected for every point given all visible viewpoints. When projecting a point to a texture image, bilinear interpolation is used to calculate the color value. Second, color outliers are removed by statistical analysis. For this, we compute the baseline color $\mathbf{C}_M = (R_M, G_M, B_M)$ as the median of all color samples for each individual color channel. The color distance between a color sample (R_i, G_i, B_i) and the baseline color is defined by

$$d_C = \max\left\{|R_i - R_M|, |G_i - G_M|, |B_i - B_M|\right\}. \tag{10.2}$$

Any color sample with d_C larger than threshold d_T is considered to be an outlier. By removing outliers, we reject most of wrong color values resulting from calibration and modeling errors. In our experiment, $d_T = 50$ yields good results. Last, the blended color is computed as a weighted average of the remaining valid color samples. We have developed a new empirical weight for the valid color sample, which is inserted in a blending function. Given N valid color samples, the blending function is defined by

$$\mathbf{C} = \frac{\sum_{i=1}^{N}(|\cos\theta_i|/d_i) \times \mathbf{C}_i}{\sum_{i=1}^{N}(|\cos\theta_i|/d_i)}, \tag{10.3}$$

where variable i denotes the ith color sample, θ_i is the angle between the point normal and camera viewpoint, d_i is the distance to the viewpoint and vector $\mathbf{C}$ represents the (R, G, B) color value. The point normal is estimated with local least-squares plane fitting. With this blending function, we assume a color sample is more reliable when it is closer to the camera and the local surface is more orthogonal to the viewpoint. Figure 10.3 (c) shows the point cloud colored with the proposed blending. In comparison to Figure 10.3 (a) and (b), a clean and smooth coloring is obtained without artificial color changes. Figure 10.3 (d) depicts a dense colored point cloud with adaptive point upsampling prior to coloring. Obviously, more appearance information is available with upsampling, yielding a more visually appealing colored model.

10.1.2.3 Multi-view Texture Generation

With the dense colored model obtained from last section, we can additionally generate multi-view textures to approximate the input color sequences and enable straight-forward texture mapping. In this process, textures are generated for each rendered depth image obtained in Section 10.1.2.1, where a color value is assigned to pixels with valid depth value. To do this, every depth pixel is back-projected by Equation (3.62) and transformed into the world coordinate by Equation (3.67) to obtain its global coordinate ${}_g$. The nearest point to $\mathbf{p}$ in the dense colored model is searched and its color is assigned tothe corresponding pixel. Figure 10.4 shows the generated texture in comparison to the original color image. It can be seen that our inexpensive color assignment yields good texture with well-preserved

Figure 10.4: Generated texture from colored point cloud on the right, in comparison to the input color image from the same viewpoint on the left.

appearance. However, due to blending of low-quality textures, imperfect calibration and errors in mesh modeling, the generated texture is less sharp and slightly distorted.

Using the generated texture, a straight-forward texture mapping can be obtained. Given the generated textures are based on a blended colored point cloud, a triangle can be mapped to any visible texture patches without producing artificial color changes. However, to provide more pictorial details for rendering, we assign a weight to the visible texture patches by $\omega_i = \alpha_i \cdot |\cos \theta_i|$, where α_i and θ_i are the area and angle between point normal and viewpoint of the ith patch, respectively. The patch with highest weight is assigned to the triangle. Figure 10.5 presents a comparison of the dense colored model and the textured model obtained from the proposed processing. It can be seen, the two colored models are highly consistently with visually appealing texturing.

10.2 Online Multi-volume RGB-D Mapping and Tracking [2]

We now consider an on-line mapping scenario. In this setting, we study the real-time RGB-D SLAM algorithms with a volumetric representation, similar to KinectFusion (**Newcombe et al.** 2011a) and Kintinuous (**Whelan et al.** 2012). With KinectFusion alike algorithms, the core (colored) TSDF volumes perform two important functionality: reconstruction and prediction. With reconstruction, multi-view measurements are integrated into a continuous

[2] © (2015) COPYRIGHT Society of Photo-Optical Instrumentation Engineers (SPIE). Textual materials, figures, tables reused with permission from Lingni Ma, Egor Bondarev and Peter H. N. de With, *Multi-volume Mapping and Tracking for Real-time RGB-D Sensing*, in SPIE Image Processing: Algorithms and Systems XIII, 2015.

Figure 10.5: Generated texture from colored point cloud on the right, in comparison to the input color image from the same viewpoint on the left.

function while being discretized by voxels. With prediction, a depth image is obtained by raycasting the TSDF volume and subsequently used to formulate geometric residual in tracking. For reconstruction purpose, smaller voxel size is important to represent details. Whereas, for robust tracking, large volume size is essential to ensure wider field of view. Given limited GPU memory, trade-offs need to be made. The objective of this section is therefore, to develop a better solution to satisfy the needs of both ends.

Usually a single voxel requires 32 bits memory to store both TSDF and color values. Given the total memory usage grows cubically with respect to the volume resolution, the amount of voxels rarely exceeds 512^3 for 2 GB GPU. As a result, a commonly used $(3\text{ m})^3$ volume gives voxels of $(5.86\text{ mm})^3$ dimension, which is insufficient to fully preserve the visual information and the fine geometrical details. To provide a better leverage between reconstruction and prediction needs, we derive at is a multi-volume solution. We now describe this technique in detail.

10.2.1 Volume Configuration

With the objective to provide detailed mapping without sacrificing the accuracy and robustness of tracking, two volumes are allocated on the GPU. We refer these two volumes as the *constructor* and the *helper*. The constructor volume is responsible for detailed mapping of the geometry and the appearance. To this end, it is configured to have a relatively small dimension and a high voxel resolution. In this thesis, it is found that the constructor of $(1.5\text{ m})^3$ dimension and 512^3 voxel resolution yields a good mapping quality. While the detailed mapping is achieved with the constructor volume, tracking however becomes prone

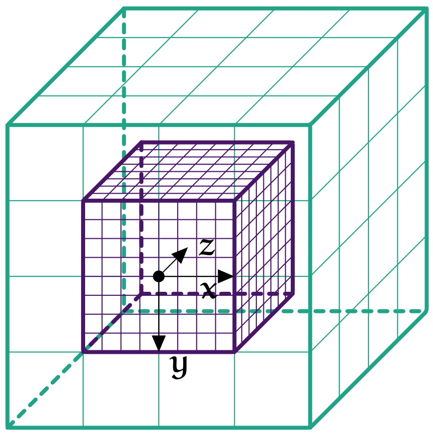
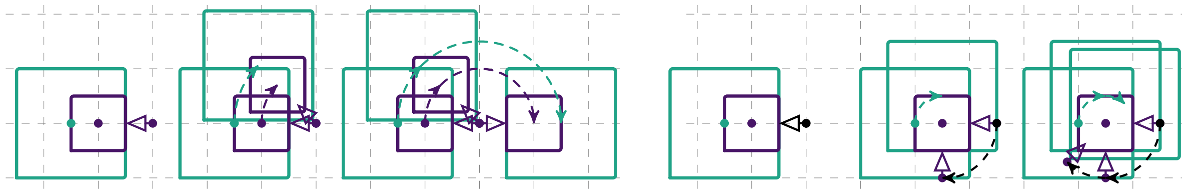


Figure 10.6: Initial position of the constructor volume (in dark color) and the helper volume (in light color). The two volumes are centered and aligned to the xy plane. The camera is placed along the z axis.



(a) the camera is fixed at one location, while rotating 360° to map its surroundings

(b) the camrea moves along a circle to map objects from all perspectives

Figure 10.7: Illustration of volume shifting with an analogical 2D plot. The constructor and the helper volumes are depicted in dark color and light color, together with the arrow to indicate the camera pose.

to failure due the limited vision. To overcome this problem, the helper volume is designed to have a large spatial dimension with a low resolution, *e.g.*, a $(3\ \text{m})^3$ volume of 256^3 voxels.

Each voxel in the constructor is associated with a 8 bit TSDF value, a 8 bit TSDF weight, a 12 bit RGB color value and a 4 bit color weight. This ensures that both depth and color frames are properly fused. In contrast to the constructor volume, only a 8 bit TSDF value and a 8 bit TSDF weight are stored by each voxel of the helper volume. Since the helper volume is used to assist in tracking, only depth frames are integrated into it. As a result, a 256^3 helper volume requires 64 MB of extra memory, which is marginal to the 1 GB memory demanded by the constructor volume. It should be noted that, even the helper volume is of 512^3 voxel resolution and requires 512 MB extra memory, the total memory demands by both volumes still fit in the memory of a modern GPU.

10.2.2 Object-Oriented Volume Shifting

In order to map unbounded space with limited GPU memory, the rolling volume technique (**Whelan et al.** 2012) is applied. In placing and shifting the volumes with respect to the camera, an object-oriented shifting approach is developed. The principle idea is to ensure good perceptive field for the camera and to maximize volume occupancy during the mapping process.

Since the constructor volume and the helper volume are used for different purposes, we shift them independently. Figure 10.6 shows the initial position of the constructor and the helper volumes. The two volumes are centered and aligned in the xy plane, while the camera is placed along the z axis pointing to the volume centers. When the camera moves, both constructor and helper volumes are adjusted accordingly. To determine how to shift the volumes, the following parameters are defined. First, we define vision center $\mathbf{V}_c$ of a camera to be a 3D point $\mathbf{V}_c = (0, 0, R)^T$ in the camera coordinates. We further define a range parameter R to be computed by $R = 0.5D_v + d$, where D_v is the volume dimension and d is a displacement factor. Consider that RGB-D camera usually have a near-field occlusion (*e.g.*, 0.8 m is typical for the Kinect camera in far-range mode), the displacement factor d can be used to compensate such occlusion by keeping the camera over d distance away from the volume. The displacement factor can also be used to adjust the camera to be of an arbitrary distance away from the volume center. In a special case with $d = -0.5D_v$, the camera is placed exactly at the volume center.

With the previous definitions, the initial vision centers of the constructor and the helper coincide with their volume centers. During camera movement, we check the current vision center of the ith frame in the volume coordinate system by computing $\mathbf{V}_c^{(i)} = \mathbf{T}_i \mathbf{V}_c$. The transformation matrix $\mathbf{T}_i$ is the estimated camera pose from the i^{th} camera coordinate to the current volume coordinate. Whenever the current vision center deviates more than d_{th} distance away from the volume center, the volume is shifted accordingly to realign both.

To further illustrates our volume shifting methods, Figure 10.7 depicts two scenarios. In the first scenario, the camera is fixed at one location, while rotating 360° to map its surroundings. Both constructor and helper volumes move approximately along a circular trajectory. In the second scenario, the camera moves along a circular trajectory around the constructor with the objects at the center. With a circle radius equal to the view range of the constructor, the constructor volume remains in a static position, whereas the helper volume will be shifted around the constructor. Notice that in both scenarios, despite the volumes are shifted independently, the helper always ensures wide vision for tracking.

The physical interpretation of this volume shifting strategy is that the volume remains its position as long as the camera moves within certain thick sphere layer around the volume and it is pointing to the volume center. Since the camera vision center is usually where the objects of interests are, this volume shifting strategy tends to optimize the voxel occupancy. We name this strategy *object-oriented volume shifting*. In contrast to the shifting strategy which places the camera at the volume center (**Whelan et al.** 2012), this shifting strategy

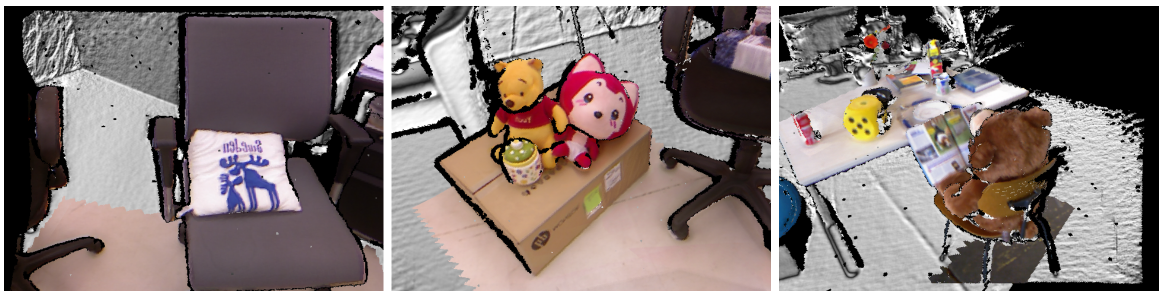


Figure 10.8: Synthetic view by ray casting the volumes. The colored and gray scale pixels are obtained from the constructor and the helper, respectively. Notice how the helper volume fills in the missing pixels to assist tracking.

enables an adjustable distance to the volume center and a more flexible choice of volume sizes. Compared to the strategy to transform volumes with arbitrary rotation and translation (**Roth and Vona** 2012), our method transforms the camera with respect to the volume and hence no interpolation is required to remap the voxel values.

10.2.2.1 Volume Fusion and Ray Casting

Following the scheme of volumetric mapping (**Curless and Levoy** 1996; **Newcombe et al.** 2011a), each pair of depth and color frames are first integrated into the volumes to update the 3D model with an implicit representation, and then a synthetic depth frame is generated by ray casting for tracking of the next frame. To perform volume fusion and ray casting, the camera poses with respect to both the constructor volume and the helper volume are required. In our algorithm, we estimate the camera pose with respect to the constructor volume. Since the constructor and the helper are always axis-aligned, the pose with respect to the helper can be calculated by concatenating the translation from the constructor volume to the helper volume.

At the data fusion stage, the TSDF and color values associated to voxels, are updated according to Equation (3.90). To compute the weight, we set $w_{i+1}(\mathbf{x}) = 10 \cdot |\cos\theta|$, where θ is the angle between the camera viewpoint and the surface normal. Consider that the depth measurements around object boundaries are often noisy, the weight of boundary pixels are set to zero and thus excluded from fusion.

At the ray casting stage, we first ray cast the constructor volume, and then fill the missing pixels by ray casting the helper volume. Since the constructor volume is of a higher resolution than the helper volume, its raycasted values are also of a higher accuracy. Figure 10.8 shows the synthetic views from ray casting. The reader should note the detailed coloring from the constructor and the wide vision coverage from the helper.

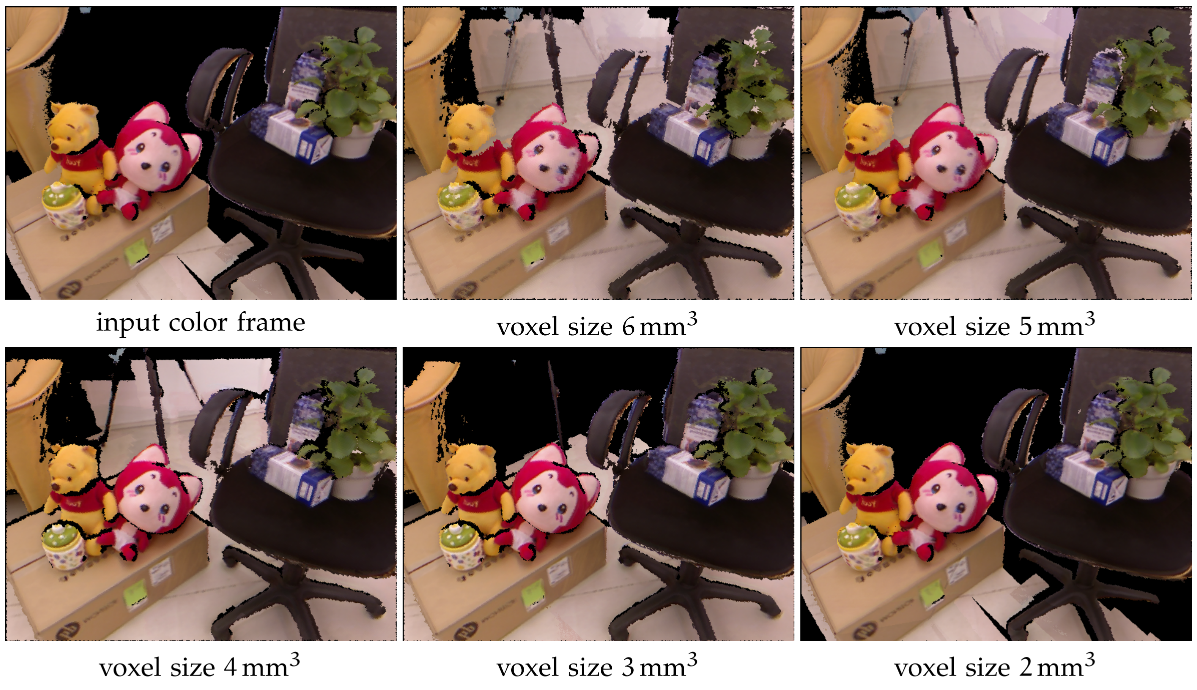


Figure 10.9: The quality of color maps generated with voxel of different sizes, in comparison to the input images.

10.2.3 Evaluations

To objectively evaluate the improvements of the visual quality by multi-volume processing, we examine the color maps generated with different voxel sizes. For this purpose, we obtain maps with a single volume of 512^3 voxel resolution and different voxel sizes. The synthetic views are rendered from the camera poses after obtaining the final mapping. Taking the corresponding input images as a reference, the Peak-Signal-to-Noise Ratio (PSNR) and the Structure Similarity Index Matrix (SSIM) of the rendered images are computed. Figure 10.9 shows the rendered images using different voxel sizes. It can be observed that the pictorial details get sharper with a smaller voxel dimension. The PSNR and the SSIM measurements of 650 frames are given in Table 10.1. The results show that the PSNR and the SSIM increase with smaller voxel and the maximum quality is achieved with $(3\text{mm})^3$ voxel size. The $(2\text{ mm})^3$ voxel does not produce the highest quality because the volume size becomes too small ($(1\text{ m})^3$ with 512^3 voxels) which leads to a dropping accuracy in tracking.

Table 10.1: Colored maps generated from voxels of different sizes. The visual quality is measured with PSNR and SSIM of rendered images.

voxel	PSNR (dB)				SSIM			
configuration	min	max	median	std	min	max	median	std
$(6\text{ mm})^3$	16.169	21.569	19.485	1.363	0.617	0.839	0.749	0.067
$(5\text{ mm})^3$	16.305	21.724	19.564	1.636	0.617	0.839	0.769	0.066
$(4\text{ mm})^3$	16.511	22.741	19.974	1.309	0.669	0.861	0.785	0.038
$(3\text{ mm})^3$	**17.371**	**24.277**	**21.096**	1.181	**0.712**	**0.876**	**0.819**	0.039
$(2\text{ mm})^3$	16.430	23.161	20.181	1.356	0.651	0.849	0.762	0.041

CHAPTER 11

RGB-D Mapping with Plane Priors

THIS chapter presents further analysis to Chapter 4 in two aspects. First, we describe the curvature-based plane detection algorithm to segment point cloud maps into planar and non-planar surfaces. Second, we discuss how to extend these algorithms for incremental and batch processing for gradually expanding point cloud maps.

11.1 Curvature-Based Plane Segmentation Algortihm [1]

Planes are characterized by their perfect flatness and can be described as sets of points that have zero curvature. In practice, open scene point cloud data can be quite noisy and points belonging to planes do not have a curvature of exactly zero. However, the curvature of points lying on planes is still low enough to distinguish them from points belonging to non-planar surfaces. This observation motivates the functionality of our algorithm, which is built upon the work of **Rabbani et al.** (2006).

11.1.1 Method

Given a dense point cloud, the curvature-based algorithm segments planar surfaces iteratively, the pseudo code of this algorithm is given in Algorithm 11.1. At each iteration, the point with the lowest curvature among the remaining unsegmented points is selected. This initiates a region growing process, where the selected point become the first seed point, and its normal is used as an estimation for the underlying plane. To grow the underlying planar surface incrementally, the k-nearest neighbors of the current seed point are determined and their normals are compared to the estimated plane normal. A neighboring point is added to the current segment if its normal is consistent with the plane normal within an angle threshold. A qualified point is also enqueue as a new seed point for further region growing if its

[1] © 2013 IEEE. Textual materials, figures, tables reused with permission from Lingni Ma, Raphale Favier, Luat Do, Egor Bondarevand Peter H. N. de With, *Plane Segmentation and Decimation of Point Clouds for 3D Environment Reconstruction*, in proc. of IEEE 10th Consumer Communications and Networking Conference (CCNC), 2013.

ALGORITHM 11.1: Curvature-based plane segmentation.

```
1  Input: 3D point cloud made of points $\mathbf{p}_i \in \mathbb{R}^3$ with normals $\mathbf{n}_i$ and curvatures $c_i$
2  Parameter: $\theta_{th}$ angle threshold, $c_{th}$ curvature threshold
3  Output: A set of planar segments
4  while points remain unsegmented or the queue is not empty do
5      if the queue is empty then
6          pick a seed point $\mathbf{p}_s$ with the lowest curvature
7          set the plane normal $\mathbf{n}_p$ to be the normal of $\mathbf{p}_s$
8      else
9          pop out a seed point $\mathbf{p}_s$ from the queue
10     mark $\mathbf{p}_s$ as segmented
11     compute the k-nearest neighbours of $\mathbf{p}_s$
12     foreach unsegmented neighbour $\mathbf{p}_i$ do
13         if $\arccos(\mathbf{n}_p, \mathbf{n}_i) < \theta_{th}$ then
14             add $\mathbf{p}_i$ to the current segment, mark $\mathbf{p}_i$ segmented if $c_i < c_{th}$ then
15                 add $\mathbf{p}_i$ to the queue
16     if the queue is empty then
17         output the current segment as a plane
```

curvature is sufficiently small. When no more points can be added to the current segment, a plane surface is produced. Afterwards, the whole process restarts with the remaining points, until the entire cloud has been processed. An early termination is reached, when the lowest curvature among the remaining points exceeds a threshold, meaning no further planar surfaces can be formed.

This curvature-based algorithm works with two parameters. The first parameter θ_{th} specifies the maximum angle between the estimated plane normal and the normal of a potential point on the plane. Typically a 10° angle works well for noisy point clouds. The second parameter c_{th} is the curvature threshold, which is used to verify whether a point should be designated as a seed point for region growing. Empirically this threshold is set to a value below 0.1.

There are two major differences between this algorithm and the method of **Rabbani et al.** (2006). The first difference is how new points are accepted into the current segment. The original algorithm always updates the normal used for the integration to be the current seed point. In our algorithm, the normal for comparison is fixed to the normal of the first seed point throughout the entire region growing process. Consider the first seed point has the lowest curvature, its normal can be assumed to be a good estimation for the normal of the entire planar segment. With this modification we avoid the detection of smoothly-connected shapes such as spheres and cylinder-like structures. The second modification concerns the estimation of point curvature. In **Rabbani et al.** (2006), curvature is estimated to be the residual of local LS plane fitting. In our algorithm, the curvature is estimated using Equation (3.92) (**Pauly et al.** 2002). The two estimations are related. Assume the principal

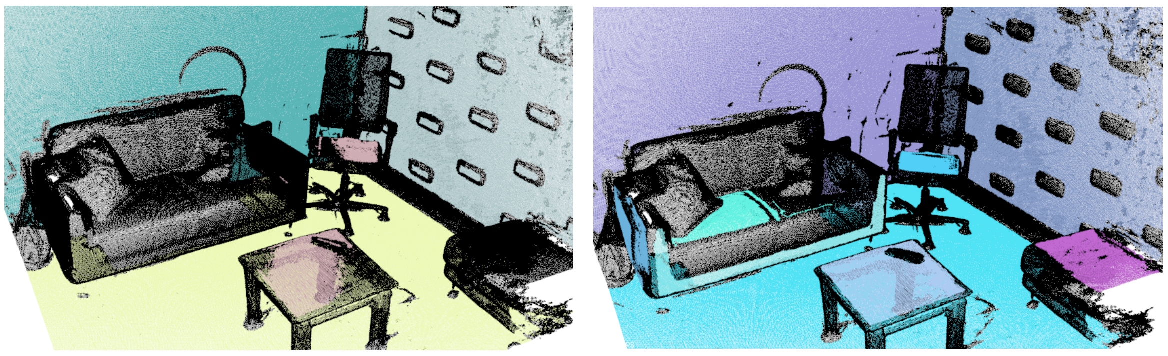


Figure 11.1: Plane detection with the RANSAC-based algorithm (left) and the curvature-based segmentation (right). Each detected plane is marked with a different color and points marked as black do not belong to any planes.

values produced by the PCA of the local points to be $\lambda_1, \lambda_2, \lambda_3$, with $\lambda_1 \leqslant \lambda_2 \leqslant \lambda_3$, the residual of local LS plane fitting is given by λ_1 as discussed in Section 3.2.3.3. Therefore, **Rabbani et al.** (2006) directly uses λ_1, whereas λ_1 is normalized by $\lambda_1 + \lambda_2 + \lambda_3$ in our estimation to be more robust.

11.1.2 Comparison to RANSAC

One common algorithm to detect planes is the RANSAC. In Figure 11.1, we give a visual comparison between our algorithm and RANSAC. In both algorithms, we accept planes with a size larger than 5% of the total points. It can be seen that planes detected by RANSAC contain fractions from different disconnected objects, which are considered to best fit a plane model. In contrast, our algorithm detects complete and continuous planes, which have a physical representation, such as the wall, the floor, the table, etc. This property enables the further processing steps for 3D reconstruction to be more reliable and reasonable.

In addition to the accuracy of plane detection, the computational efficiency of our algorithm is also examined. In Table 11.1, the execution times of the curvature-based algorithm and RANSAC are compared. To ensure satisfying results, 100 iterations are used for RANSAC. The comparison shows that the curvature-based algorithm is on average 2 to 7 times faster in execution than RANSAC. The computational efficiency is especially beneficial for large point clouds. For experiment, the algorithm is implemented on Ubuntu 11.10 on an Intel Core i5 CPU 750 of 2.67GHz with 3.8GB memory.

Table 11.1: Comparison of the computational efficiency for the RANSAC-based algorithm and our algorithm. The execution time is measured in second.

num. points	RANSAC (s)	our algorithm (s)
74,756	1.35	0.63
288,011	9.60	2.66
650,284	12.22	5.82
1,174,824	20.96	10.30
2,311,672	159.69	20.94

11.2 Incremental Planar Simplification for Gradually Expanding Maps[2]

Many RGB-D SLAM algorithms produce dense point cloud maps incrementally in real-time, *e.g.*, Kintinuous from **Whelan et al.** (2015b). In this section, we extend the algorithms detailed in Chapter 4 and the plane segmentation method in Section 11.1 to batch process incrementally expanding point cloud maps.

11.2.1 Segment Merging and Growing

Our method involves maintaining a pool of unsegmented points which are either segmented as new planes, added to existing planes or deemed to not belong to any planar segment. Firstly we define a distance-based plane merging method that determines whether or not to merge two planar segments based on the distance between the points in each segment. We list this as Algorithm 11.2 and henceforth refer to it as the $mergePlanes$ method.

Assuming the input to our system is a small part of a larger point cloud map that is being built up over time we must define a method for growing existing planar segments that were found in our map in the previous timestep of data acquisition. We maintain a persistent pool of unsegmented points M where each point $M_i \in \mathbb{R}^3$ and also contains a timestep value M_{i_t}, initially set to zero. When a new set of points are added to the map, they are added to the set M, which is then sorted by the curvature of each point. A batch segmentation of M is then performed (as described in Chapter 11), producing a set of newly segmented planes N. From here we perform Algorithm 11.3, which will grow any existing segments and also populate the set S, that maintains a list of planes which are similar in orientation but not close in space. Algorithm 11.4 lists the method for merging similar planes that eventually grow close enough in space to be merged together.

Each time new data is added to the map Algorithm 11.3 and Algorithm 11.4 are run, after which the timesteps values of all remaining unsegmented points in M are incremented by one. Points with a timestep value above a specified threshold are removed from the point

[2] © 2014 Elsevier B.V.. Textual materials, figures, tables reused with permission from Thomas Whelan, Lingni Ma, Egor Bondarev, Peter H. N. de Withand John McDonald, *Incremental and Batch Planar Simplification of Dense Point Cloud Maps*, in Robotics Autonomous Systems, July 2015.

ALGORITHM 11.2: Method for merging two planar segments.

```
Input: A planar segment with normal $A_n$ and point cloud $A_C$
       B planar segment with normal $B_n$ and point cloud $B_C$
       $B_H$ concave hull of B
Parameter: $d_{th}$ distance threshold
Output: True or False if segments were merged or not
foreach point $h_i$ in $B_H$ do
    if $\exists A_{C_k}$ s.t. $\|h_i - A_{C_k}\|_2 < d_{th}$ then
        $A_n \leftarrow (A_n|A_C| + B_n|B_C|)/(|A_C| + |B_C|)$
        $A_t \leftarrow 0$
        append $B_C$ to $A_C$
        compute KD-tree of $A_C$
        return True
return False
```

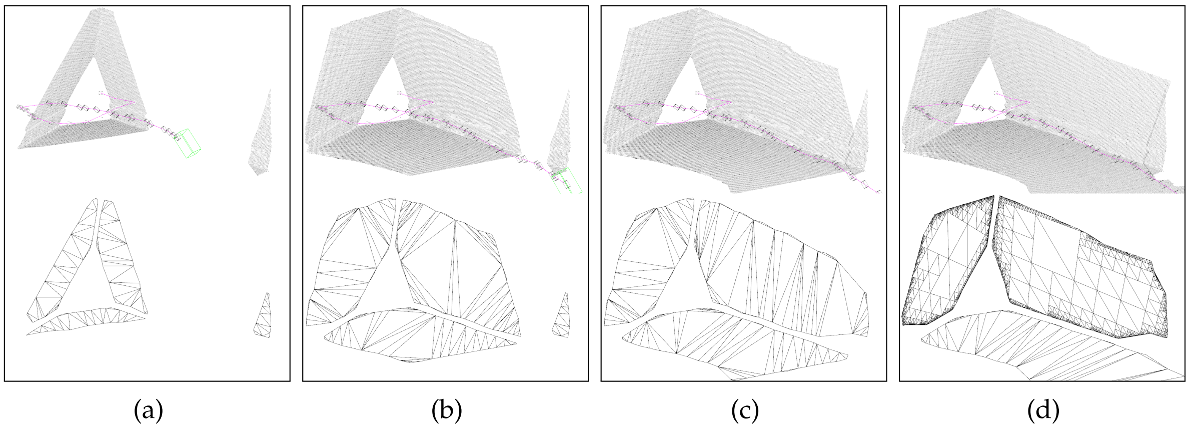


Figure 11.2: Incremental planar segmentation shown with point cloud and camera trajectory (in pink) shown above and resulting planar segments below. From left to right: (a) initially there are four segments extracted from the point cloud; (b) as the camera moves and more points are provided, the three segments on the left are grown (as described in Algorithm 11.3); (c) the upper-right most segment grows large enough to be merged with the small segment on the right (as in Algorithm 11.4); (d) once the camera has moved far enough away from the two upper segments they are finalised.

pool and marked as non-planar. Algorithm 11.3 will add new segments to the map, grow recently changed segments and ensure similar planes that have the potential to grow into each other are kept track of. By including both spatial and temporal thresholds it ensures that the process scales well over time and space. Algorithm 11.4 merges segments which may not have initially been close together in space but have grown near to each other over time. Figure 11.2 shows an example of the incremental planar segmentation process in action.

ALGORITHM 11.3: Method for growing planar segments.

```
1  Input: N set of new planar segments with normals $N_{i_n}$, point clouds $N_{i_C}$ and timesteps $N_{i_t}$
2         Q set of existing planar segments with normals $Q_{i_n}$, point clouds $Q_{i_C}$ and timesteps $Q_{i_t}$
3         $C_t$ current position of sensor producing the map
4  Parameters: $n_{th}$ normal merge threshold, $t_{th}$ timestep threshold, $td_{th}$ timestep distance
     threshold
5  Output: S set of similar but non-merged segments
6  foreach newly segmented plane $N_i$ do
7      $R \leftarrow \varnothing$
8      $\mathtt{gotPlane} \leftarrow \text{False}$
9      foreach existing plane $Q_i$ do
10         if $!Q_{i_{finalised}}$ and $\arccos(N_{i_n}, Q_{i_n}) < n_{th}$ then
11             compute concave hull H of $N_i$
12             $\mathtt{gotPlane} \leftarrow \mathtt{mergePlanes}(Q_i, N_i, H)$
13             if gotPlane then
14                 break
15             else
16                 add $Q_i$ to R
17     if !gotPlane then
18         compute KD-tree of $N_{i_C}$
19         $N_{i_t} \leftarrow 0$
20         add $N_i$ to Q
21         foreach similar plane $R_i$ do
22             add $(N_i, R_i)$ tuple to S
23     remove all points $N_{i_C}$ from M
24 foreach existing plane $Q_i$ do
25     $Q_{i_t} \leftarrow Q_{i_t} + 1$
26     if $Q_{i_t} > t_{th}$ and $\forall q \in Q_{i_C}, \|C_t - q\|_2 > td_{th}$ then
27         $Q_{i_{finalised}} \leftarrow \text{True}$
28 foreach similar plane $S_i$ do
29     if $S_{i_1 finalised}$ or $S_{i_2 finalised}$ then
30         delete $S_i$
```

11.2.2 Evaluation of the Incremental Segmentation

To evaluate the proposed algorithms, we ran the C++ implementation on Ubuntu Linux 12.04 with an Intel Core i7-3930K CPU at 3.20 GHz with 16 GB of RAM. The same data as shown in Figure 4.10 are used to compare the batch segmentations to the incremental segmentations qualitatively and quantitatively. We use the open source software CloudCompare[3] to align the batch and incremental models of each dataset together to compute statistics. We quantify the quality of the incremental segmentation versus the batch segmentation by using the “cloud/mesh" distance metric provided by CloudCompare. The process

[3] http://www.danielgm.net/cc/

ALGORITHM 11.4: Merging segments that have grown closer.

```
Input: S set of similar but non-merged segments with point clouds $S_{i_C}$ and alpha values $S_{i_\alpha}$
        Q set of existing planar segments
Output: S set of merged segments
foreach similar plane $S_i$ do
    gotPlane ← False
    if $|S_{i_{1C}}| > |S_{i_{2C}}|$ then
        swap $S_{i_1}$ and $S_{i_2}$
    if $S_{i_{1\alpha}} != |S_{i_{1C}}|$ then
        compute concave hull $S_{i_{1H}}$ of $S_{i_1}$
        $S_{i_{1\alpha}} \leftarrow |S_{i_{1C}}|$
    gotPlane ← mergePlanes($S_{i_2}, S_{i_1}, S_{i_{1H}}$)
    if gotPlane then
        foreach existing plane $Q_i$ do
            if $Q_i == S_{i_1}$ then
                delete $Q_i$
                break
        foreach similar plane $S_j$ do
            if $S_i == S_j$ then
                continue
            if $S_{j_1} == S_{i_1}$ then
                $S_{j_1} \leftarrow S_{i_2}$
            else if $S_{j_2} == S_{i_1}$ then
                $S_{j_2} \leftarrow S_{i_2}$
            if $S_{j_1} == S_{j_2}$ then
                delete $S_j$
        delete $S_i$
```

involves densely sampling the batch planar model mesh to create a point cloud model which the incremental model is finely aligned to using ICP (**Besl and McKay** 1992). Then, for each vertex in the incremental planar model, the closest triangle in the batch model is located and the perpendicular distance between the vertex and closest triangle is recorded. Five standard statistics are computed over the distances for all vertices in the incremental model: Mean, Median, Standard Deviation, Min and Max. These are listed for all four datasets in Table 11.2. Figure 11.3 shows heatmap renderings of the "cloud/mesh" error of each incremental segmentation compared to the batch segmentation. Notably in each dataset there are a number of highlighted green planes, these are planes which were not detected in the incremental segmentation model but exist in the batch segmentation. In general the incremental segmentation occasionally fails to segment small planar segments whereas the batch segmentation always finds all planes that match the criterion set out in Algorithm 11.1. Additionally, as the incrementally grown planes use a moving average for the planar segment normal, some planes may have a slightly different orientation when compared to the batch model. This is evident in particular in Figure 11.3 (a) and (c). Taking this qualitative

Table 11.2: Incremental versus batch planar segmentation statistics. All values shown are in meters, on the distances between all vertices in the incrementally segmented model and the nearest triangles in the batch segmented model.

Dataset	1	2	3	4
Mean	0.020	0.038	0.111	0.028
Median	0.015	0.004	0.015	0.010
Std.	0.021	0.108	0.251	0.065
Min	0.000	0.000	0.000	0.000
Max	0.157	0.823	1.389	0.719

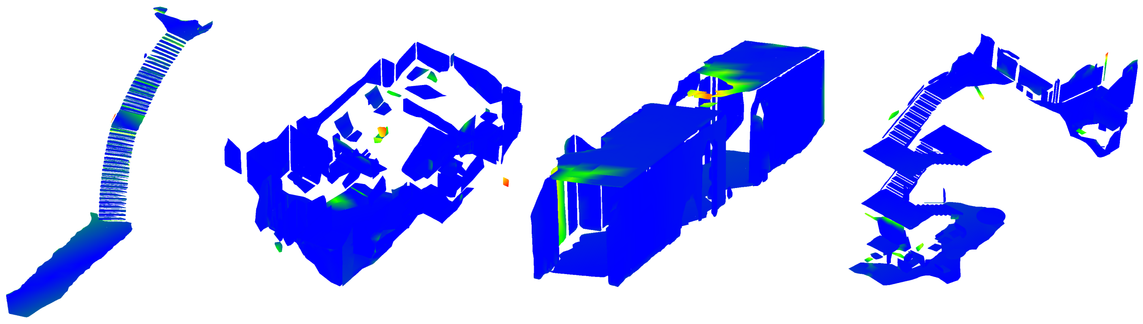


Figure 11.3: Heat maps based on the distances between all vertices in the incrementally segmented model and the nearest triangles in the batch segmented model for all four datasets. Colour coding is relative to the error obtained where blue is zero and tends towards green and red as the error increases. This figure is best viewed in full colour. All values shown are in meters.

information into account as well as the statistics in Table 11.2 we find that the incremental segmentation algorithm produces segmentations extremely close to what would be achieved using the batch process and is suitable to use in a real-time system that must generate and use the planar model online.

CHAPTER 12

Semantic RGB-D Mapping

In Chapter 7, we discussed training CNN with multi-view constrains to obtain consistent semantic segmentation on RGB-D sequences. Here, we provide further details on CNN training and describe the method to fuse a sequence of semantic segmentations of individual RGB-D images into semantic 3D maps.

12.1 Efficiency with Deep Supervision

In Chapter 7, we formulate CNN training with multi-scale loss minimization. In this ablation study, we analyze the benefit of multiscale loss in guiding decoder with the upsampling. To this end, we take FuseNet-SF1 (**Hazirbas et al.** 2017) as the baseline and incrementally add extra loss to penalize the training at different output resolution from fine to coarse. We denote the different configurations by MSLM-loss$x$, where x refers to the number of output scale contributes to the total loss. Hence, the baseline FuseNet-SF1 is trained with loss penalization only at the finest resolution, whereas MSLM-loss5 is trained with penalization on all scales. The obtained segmentation accuracy is compared in Table 12.1. As the results demonstrate, adding additional loss increases the performance consistently. This indicates that penalizing the low-resolution prediction guides the upsampling process in the decoder. In Figure 12.1, we also show the loss curve of these different configurations obtained from training with the same hyper-parameters. Interestingly, adding multi-scale loss penalizations speeds up the training process with a much better convergence rate. With MSLM-loss5, the network converges significantly faster than the baseline. The loss curves also show that training is more stable with multi-scale loss minimization where less oscillation is observed. These results indicate that MSLM potentially relaxed the original non-convex minimization problem into a better convex one.

In the second study, we visualize the uncertainty of the network predictions in Figure 12.2. To obtain the visualization, we show the probability of the most likely class prediction. It can be seen that CNN predictions are very certain inside object, where uncertainty only increases approaching the segmentation boundaries. On interesting application of the uncertainty

Table 12.1: Semantic segmentation accuracy of our network with different level of loss for NYUv2 13-class semantic segmentation tasks. We take the FuseNet-SF1 (**Hazirbas et al.** 2017) as the baseline and add loss terms successively to guide the upsampling process at different scales.

network variant	pixelwise accuracy	classwise accuracy	average IoU
FuseNet-SF1	74.46	64.90	52.06
MSLM-loss2	74.96	64.67	52.48
MSLM-loss3	75.47	66.19	53.42
MSLM-loss4	75.78	67.19	54.13
MSLM-loss5	**76.07**	**67.55**	**54.55**

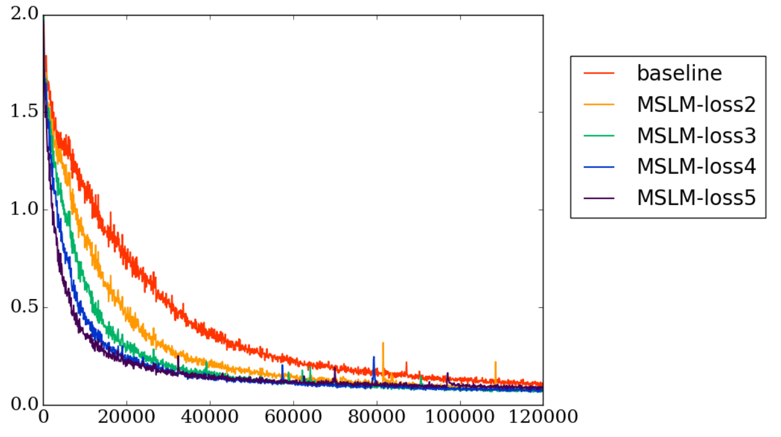


Figure 12.1: Training curves for network with different level of loss penalization. With multiscale loss minimization, training converges much faster and with less oscillations, which suggests the proposed MSLM possibly to relax the overall loss minimization into a better convex problem.

map is therefore to obtain instance information.

12.2 Semantic Mapping with Dense CRFs

To obtain semantic maps from the sequence of individual predictions, **Hermans et al.** (2014) proposed to construct a fully connected CRFs (**Krähenbühl and Koltun** 2011) from all available pixels for spatial smoothing, and then fuse the de-noised predictions with Bayesian fusion into consistent semantic maps. To obtain the unaries for CRF, **Hermans et al.** (2014) trains random forests. In our work, we experiment with a similar pipeline. Consider a keyframe-based RGB-D SLAM, we construct the semantic maps with a two-stage fusion and smoothing.

At the first stage, we smooth the keyframe prediction using the predictions from neighbor-

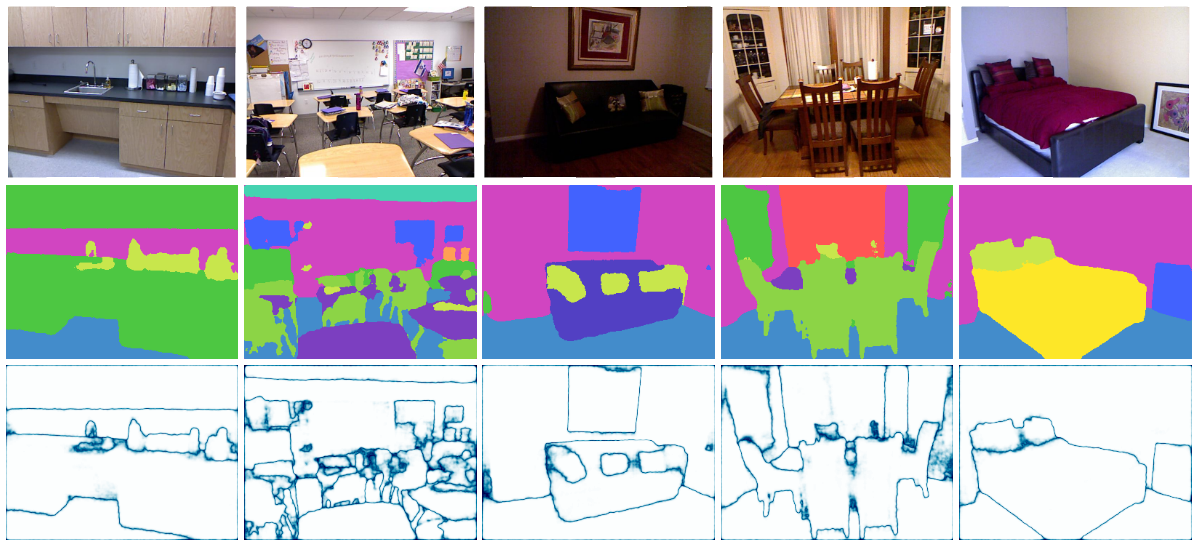


Figure 12.2: The uncertainty of CNNs semantic segmentation prediction (color code: the uncertainty increases from low to high corresponds to color from white to dark).

ing frames. With the known transformations from SLAM, the pixelwise classification score is warped into the keyframe and fused together with Bayesian update. The same processing technique developed in Section 7.4.3 is applied.

In the second stage, we apply spatial smoothing for all the keyframes with a dense CRF. To this end, we construct a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, where each node $\mathcal{V}_i$ is a pixel $\mathbf{x}_i$ from a keyframe and edges are densely connected $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. With every pixel connected to every other pixel, the CRF thus imposes label smoothing over the whole map. Let each pixel $\mathbf{x}_i$ takes a possible labeling $\mathbf{y}_i \in \mathbb{R}^K$, the CRF graph defines the Gibbs distribution that encodes the following conditional probability

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\big(-E(\mathbf{y},\mathbf{x})\big) , \tag{12.1}$$

where, $Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}} \exp\big(-E(\mathbf{y},\mathbf{x})\big)$ is known as the partition function for normalization. The Gibbs energy $E(\mathbf{y},\mathbf{x})$ can be further expressed with the unary potential E_i and the pairwise potential E_{ij} by,

$$E(\mathbf{y},\mathbf{x}) = \sum_{i \in \mathcal{V}} E_i(\mathbf{y}_i;\mathbf{x}_i) + \sum_{ij \in \mathcal{E}} E_{ij}(\mathbf{y}_i,\mathbf{y}_j;\mathbf{x}_i,\mathbf{x}_j) . \tag{12.2}$$

We define the unary potential given the CNN prediction,

$$E_i(y_i = c_i, \mathbf{x}_i) = -\log p_i(\mathbf{x}_i) \ , \tag{12.3}$$

where the probability $p_i(\mathbf{x}_i)$ of pixel x_i having label j is defined in Equation (7.1). Following **Krähenbühl and Koltun** (2011), the pairwise energy is defined to be Gaussian potentials.

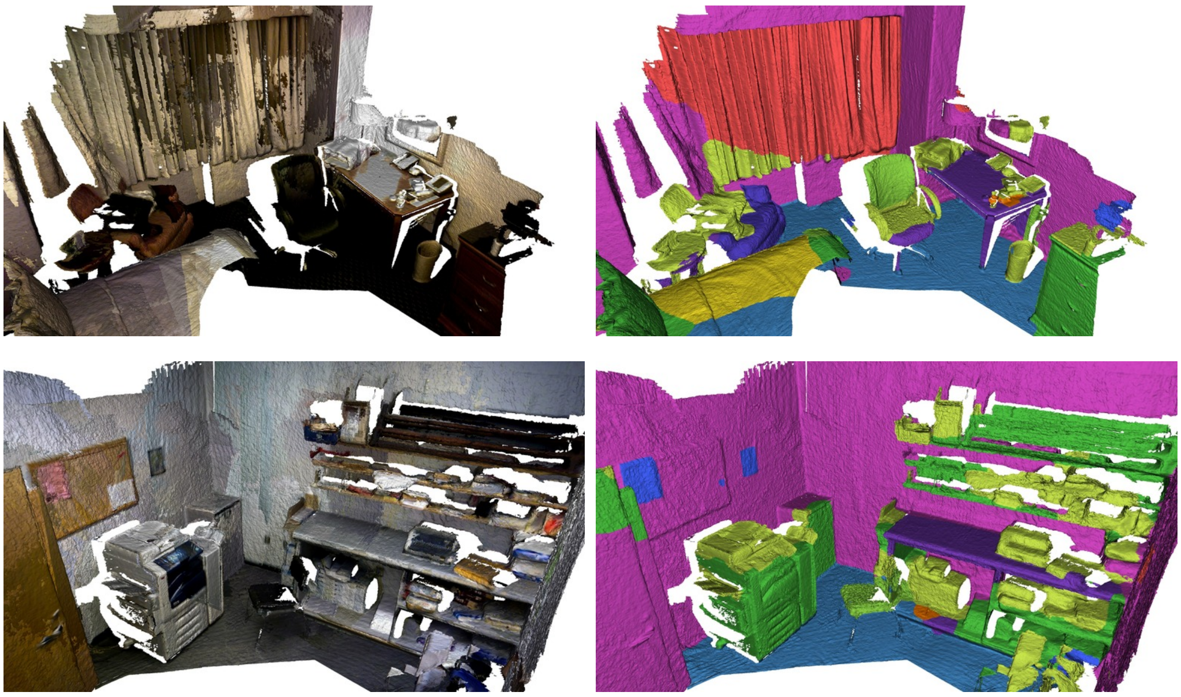


Figure 12.3: Two example semantic 3D maps obtained with the complete processing pipeline with our methods. For visualization references, we also show the colored meshes.

Since each pixel $_i$ in RGB-D can be back-projected to a colored 3D point, we denote the 3D coordinates by $\boldsymbol{\alpha}_i \in \mathbb{R}^3$ and the color value by $\boldsymbol{\beta}_i \in \mathbb{R}^3$, and define pairwise to be

$$E_{ij}(y_i, y_j) = [\![y_i \neq y_j]\!] \omega \exp\left(-\frac{|\boldsymbol{\alpha}_i - \boldsymbol{\alpha}_j|^2}{2\theta_\alpha^2} - \frac{|\boldsymbol{\beta}_i - \boldsymbol{\beta}_j|^2}{2\theta_\beta^2} \right) \quad . \tag{12.4}$$

The parameter ω is the weighting factor for the pairwise term, and $\theta_\alpha, \theta_\beta$ are the standard deviation for the 3D position and color, respectively. We use the Lab color space as **Hermans et al.** (2014) suggests. The inference of Equation (12.2) is solved via mean-field approximation. The naive mean-field approximation assumes all nodes are independent in the graph, and attempts to find a Gibbs distribution (Equation (12.1)) of the new graph, that minimizes the KL-divergence between the Gibbs distribution of the original graph. With Gaussian pairwise potential, this approximation can be efficiently achieved with iterative convolution (**Krähenbühl and Koltun** 2011).

The mean-field approximation performs a MAP inference, and outputs an optimal label for each node. To obtain the final semantic mapping, the keyframes are fused given an volumetric representation, where voxels are labeled by voting. When extracting meshes from the TSDF volume, the vertices then inherit labels from the corresponding voxel. Figure 12.3 present the semantic maps obtained by the proposed processing pipeline with the same color coding used in Table 7.3.

PART IV

Closure

After all, tomorrow is another day.

– Scarlett O'Hara
"Gone with the Wind"
by Margaret M. Mitchell

CHAPTER 13

Conclusions and Future Works

This thesis studied the problem of semantic mapping and tracking with RGB-D data. To this end, we have focused on algorithms to extract plane prior and integrate this information into multiple aspects of dense mapping, camera tracking and global consistency optimization. Additionally, we developed several deep learning algorithms to obtain higher-level semantic understandings from RGB-D vision. Last, we explored a human-in-the-loop annotation algorithm to obtain large-scale semantic ground truth for both 3D meshes and 2D RGB-D videos. This chapter we summary the individual conclusions from the major chapters and discuss the future research directions.

13.1 Thesis Summary

13.1.1 Dense RGB-D Mapping

In Chapter 10 (*Dense RGB-D Reconstruction*), we explored algorithms to obtain photo-realistic dense 3D reconstructions from RGB-D sequences. Given a coarse mesh and with the goal to store all multi-view textural information by surface points, we developed algorithms to adaptively upsample the points and perform color blending based on points. The dense colored colored point cloud was constrained by the input mesh, which enabled texture generation to obtained blended textures for textured model rendering. The experimental results showed that our algorithms produced the visually appealing high-resolution 3D reconstruction, while being robust to various noise from the data acquisition.

Furthermore, we introduced a novel multi-volume RGB-D SLAM algorithm. The method deployed one small volume of high voxel resolution to obtain detailed maps of near-field objects, and utilized another large volume of low voxel resolution to increase the robustness of tracking by including far-field scenes. The experimental results showed that our multi-volume processing scheme achieved an objective quality gain of 2 dB in PSNR and 0.2 in SSIM. The proposed approach was capable of real-time sensing with approximately 30 fps and can be implemented on a modern GPU.

Dense Mapping and Tracking with Plane Priors

In Chapter 4 (*Planar Simplification and Texturing of Dense Point Cloud Maps*), we focused on efficient reconstruction of dense point clouds with plane priors and proposed a hybrid mapping algorithm to process planar and non-planar surfaces separately. In this work, we first developed an efficient segmentation algorithm to detect planar surfaces as detailed in Chapter 11 (*RGB-D Mapping with Plane Priors*). For planar surfaces, we introduced a quadtree-based decimation method, which removed approximately 90% of the redundant input planar points. We further proposed two triangulation techniques to produce a compact mesh reconstruction of the decimated planar surfaces. Our methods yielded the triangulation with no more than 10% of the amount of triangles required without decimation. In addition, we developed a computationally inexpensive algorithm to automatically generate high-quality textures for simplified planar surfaces, which preserved the visual context embedded in the original dense colored point clouds. We demonstrated with a parallel multi-threaded implementation, the algorithm enhanced the processing efficiency particularly for large-scale indoor datasets. The experimental results showed that the textured models produced by the algorithm were compact, geometrically accurate and visually appealing, which were also in a useful format for many robotic applications. Further in Chapter 11, we introduced a computationally feasible and strong performing extension of the aforementioned algorithm to incrementally process gradually expanding point cloud maps.

Moving beyond mapping, in Chapter 5 (*CPA-SLAM: Consistent Plane-Model Alignment for Direct RGB-D SLAM*) we integrated plane priors into a full real-time RGB-D SLAM algorithm, and proposed the CPA-SLAM algorithm. In this work, we represented the global model with planar map instead of dense mapping. The algorithm detected individual planes from the depth image of keyframes, properly associated the keyframe observations in the global map and optimized the map in a global optimization. Based on the planar map representation, a novel tracking method was proposed to combine the direct image alignment and the global model alignment into a probabilistic formation. Consequently, our method tracked camera motion towards the nearest keyframe and the global plane model in an expectation-maximization framework. This reduced per-frame tracking drift and established additional constraints among non-overlapping keyframes once a common plane was observed. The keyframe poses and the plane model were optimized in one graph concurrently. The CPA-SLAM algorithm exhibited the state-of-the-art accuracy on public benchmarks and was capable of real-time performance.

Learning Semantics from RGB-D Vision

Going beyond plane priors, this thesis also investigated semantic scene understanding with deep CNNs. As a fundamental work for single-view RGB-D learning, Chapter 6 (*FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture*) proposed a novel fusion-based CNN network, namely FuseNet. Based on the observation that visual and depth features usually complemented each other, FuseNet exploited two network branches to filter the color and the depth images separately, while consistently combined the learned depth features into the color branch via fusion layer as the network goes deeper. Shown with experiments, our approach outperformed the existing CNN-based networks on the challenging RGB-D semantic segmentation benchmark. For ablation studies, we investigated two possible fusion strategies, *i.e.,* dense fusion and sparse fusion. The experimental results indicated that multi-stage sparse fusion yields better performance. We also showed that a single-stage fusion already produced better performance in comparison to learning from a directly stacked four-channel RGB-D input or a comprehensive depth representation HHA. We remark that a straight-forward extension of the proposed approach can be applied for other classification tasks such as image or scene classification.

Following the work on FuseNet, we further extended it to learn consistent semantic mapping from multi-view RGB-D data. Discussed in Chapter 7 (*Multi-View Deep Learning for Consistent Semantic Mapping with RGB-D Cameras*), we introduced several multi-view consistency regularizations into network training. Using the camera trajectories obtained by RGB-D SLAM algorithm, our method warped the network outputs of multiple viewpoints into a common reference view (in correspondence with the keyframe in SLAM settings) at different stages to enforce invariant feature learning under viewpoint changes. We demonstrated the superior performance of multi-view consistency training and the Bayesian fusion on the NYUDv2 13-class and 40-class semantic segmentation benchmark. All multi-view consistency training approaches outperformed the single-view trained baselines. They were key to boosting segmentation performance when fusing network predictions from multiple view points during testing. On NYUDv2, our methods set a new state-of-the-art performance using an end-to-end trained network for single-view predictions as well as multi-view fused semantic segmentation without further post-processing stages such as dense CRFs. With additional analysis in Chapter 12 (*Semantic RGB-D Mapping*), we demonstrated the semantic RGB-D mapping by fusing multi-view semantic segmentations from RGB-D sequences in a probabilistic way. Additionally, we showed the importance of multi-resolution deep supervision in boost the training performance.

In Chapter 8 (*Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform*), we studied the problem of training CNNs for detailed semantic segmentation, or in a more general concept the dense pixelwise predictions. Motivated by the structural analogy between multi-resolution wavelet analysis and the pooling/unpooling layers in CNNs, we introduced the discrete wavelet transform (DWT) into encoder-decoder CNNs and developed a wavelet-based network architecture, namely WCNN. In the proposed network, DWT is used to decompose the feature maps into frequency bands when downsampling is performed. The high-frequency components are later used at the unpooling in the decoder together with the coarse feature maps to achieve upscaling by the inverse DWT. We also proposed two wavelet-based pyramids to bridge the encoder and decoder with global contextual features. These pyramids built a multi-resolution DWT to successively reduce the spatial resolution and increase the receptive field to the entire input. Experimented with the Cityscape dataset, we showed that the proposed WCNN yield improvements in dense prediction accuracy systematically and achieved the state-of-the-art performance for semantic segmentation.

Towards Large-scale RGB-D Semantic Ground-truth Dataset

Consider the importance of high-quality large-scale RGB-D semantic ground-truth datasets, in Chapter 9 (*Human-in-the-loop Annotation for Large-scale 3D Semantic Datasets*), we proposed a close-loop annotation algorithm. At the annotation stage, we developed a unified semantic instance labeling tool to perform segmentation-aided free-form annotation over 3D meshes, as well as to inpaint missing surface reconstructions with human aid. We further rendered the labeled 3D mesh into RGB-D sequences to obtain fully annotated video and optimized the rendered labels with the joint bilateral filtering to respect object boundaries in 2D images. As shown in experiments, the proposed method produced detailed annotations with high consistency between 3D meshes and 2D videos. We further developed a close-loop labeling scheme for large-scale dataset collection. With the objective to alleviate human annotation effort, the existing annotations were used to train instance segmentation CNNs. We then developed algorithms to fuse the 2D instance predictions into the 3D mesh given the geometric cues. This process closed the annotation loop, where the ground truth data were used to train and improve an object detection model, which subsequently, generated the initial annotation (for human annotators to refine) on new data sequences. The proposed approach was evaluated by training an indoor dataset of 20 scenes, which generated automatic semantic annotation for a large office space of approximately 2500m^2 where the majority of surfaces were corrected labeled at the instance level.

13.2 Future Research

Following, we discuss several interesting future research directions.

Learning from Sequences: Multi-view Geometry and Temporal Consistency

Despite the impressive development in deep learning, it is less explored how to train deep neural networks with sequential vision data. With our work multi-view CNNs for semantic segmentation, we already see that using the scene geometry and temporal consistency embedded in multi-view sequences, learning can be better regularized and supervision signals can also be relaxed. One interesting questions hence is, how to combine and take inspirations from the classic geometry modeling methods and SLAM algorithms to improve deep neural networks. It is desired to design efficient network architecture to process sequential image data at both training and inference time. One possible direction is to consider the recurrent neural networks (RNNs) or LSTMs.

Online Semantic SLAM

With respect to SLAM algorithms, the problem of how to integrate semantics into a full online SLAM system is far from solved. With the CPA-SLAM algorithm, we proposed one possible solution to formulate SLAM with plane priors. One straight-forward extension is to integrate other geometric priors. But, such models will have limited representation capability and cannot infer the actual semantics about the observations. The work of SLAM++ (**Salas-Moreno et al.** 2013) provides another semantic SLAM formulation, however, predefined templates are required. With the impressive development of deep learning to obtain semantic understandings, the question is how to interact the building blocks of SLAM algorithms with deep neural networks towards a semantic SLAM formulation. In development of such solutions, one needs to consider the proper design of the network. Some questions of concerns are, whether instance semantic segmentation is preferred over pixelwise semantic segmentation, whether networks should only aim for semantic understandings or target at multi-tasks learning to include depth prediction, normal estimation, feature detection and ultimately towards end-to-end motion estimation. Another interesting direction to explore is how to integrate the feedback from the SLAM algorithms to improve learning, considering the fact that the conflicting predictions can be easily detected with SLAM algorithms.

Deep Learning on 3D Data

In this thesis, we perform deep learning with 2D image data. However, RGB-D data is 2.5D and geometry representations are naturally in 3D. Recently, several attempts have been made to derive efficient geometric learning. The volumetric 3D learning (**Dai et al.** 2017c; **Song et al.** 2017; **Wu et al.** 2015) is a natural extension of 2D convolutional neural networks. However, such methods is limited by the high memory consumption and thus the current volumetric learning is only feasible with low resolution predictions. Some promising solutions are the recently developed using the geometric deep learning on non-euclidean manifolds techniques (**Bronstein et al.** 2017; **Masci et al.** 2015; **Monti et al.** 2017). It is interesting to incorporate such geometric learning techniques into semantic mapping and tracking.

Large-scale RGB-D Semantic Benchmark

Despite many large-scale semantic 3D dataset are available (**Chang et al.** 2017; **Dai et al.** 2017a; **Hua et al.** 2016), there is no large-scale dataset that provides all the desired properties including, accurate complete 3D reconstruction, 2D video sequences with (approximately) error-free pose estimation, semantic instance ground-truth annotations that are consistent between 3D and 2D. Such ground-truth dataset will enable many learning possibilities. In addition to the classical tasks as semantic segmentation, depth estimation, motion estimation, it will also allow researchers to develop and evaluate problems including, the aforementioned learning from sequential data, geometric deep learning for indoor scene parsing, navigation planning, semantic SLAM etc. To this end, our proposed annotation pipeline sets up the framework for close-loop scheme. For future development, some important concerns are better shape completion strategy, more robust de-noising methods for labeling rendering, online automatic annotation suggestions and better bootstrapping algorithms to learn the initial annotation.

PART V

Appendix

You must see with eyes unclouded by hate. See the good in that which is evil, and the evil in that which is good. Pledge yourself to neither side, but vow instead to preserve the balance that exists between the two.

– Hayao Miyazaki

Detailed Contribution Disclaimer

This chapter contains the detailed disclaimer for the research publications that are included in this cumulative content (Part II), or contribute to the supplementary analysis (Part III). Note a complete list of our publications is provided in the bibliography.

Publications Included in Cumulative Content (Part II)

Planar Simplification and Texturing of Dense Point Cloud Maps. Lingni Ma, Thomas Whelan, Egor Bondarev, Peter H. N. de With and John McDonald. In proc. of European Conference on Mobile Robots (ECMR), Barcelona, Spain, 2013, pp. 164-171. ISBN: 978-1-4799-0263-7, DOI: 10.1109/ECMR.2013.6698837.

In Chapter 4

Authors	Lingni Ma[1]	lingni.ma@in.tum.de
	Thomas Whelan[2]	twhelan@oculus.com
	Egor Bondarev[1]	e.bondarev@tue.nl
	Peter H. N. de With[1]	p.h.h.de.with@tue.nl
	John McDonald[2]	johnmcd@cs.nuim.ie
	[1] Technische Universiteit Eindhoven	
	[2] National University of Ireland Maynooth	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	© 2013 IEEE	Reprinted with permission from IEEE

CPA-SLAM: Consistent Plane-model Alignment for Direct RGB-D SLAM. Lingni Ma, Christian Kerl, Jörg Stückler and Daniel Cremers. In proc. of IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016, pp. 1285-1291. ISBN: 978-1-4673-8026-3, DOI: 10.1109/ICRA.2016.7487260.

In Chapter 5

Authors	Lingni Ma[1]	lingni.ma@in.tum.de
	Christian Kerl[1]	kerl@in.tum.de
	Jörg Stückler[1]	joerg.stueckler@tuebingen.mpg.de
	Daniel Cremers[1]	cremers@in.tum.de
	[1] Technische Universität München	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	© 2016 IEEE	Reprinted with permission from IEEE

FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture. Caner Hazirbas, Lingni Ma, Csaba Domokos and Daniel Cremers. In proc. of Asian Conference on Computer Vision (ACCV), Springer, Cham, 2016, pp. 213-228. ISBN: 978-3-319-54181-5, DOI: 10.1007/978-3-319-54181-5_14.

In Chapter 6

Authors	Caner Hazirbas[1]	hazirbas@in.tum.edu
	Lingni Ma[1]	lingni.ma@in.tum.de
	Csaba Domokos[1]	csaba.domokos@in.tum.de
	Daniel Cremers[1]	cremers@in.tum.de
	[1] Technische Universität München	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	© 2016 Springer	Reprinted with permission from Springer

Multi-view Deep Learning for Consistent Semantic Mapping with RGB-D Cameras. Lingni Ma, Jörg Stückler, Christian Kerl and Daniel Cremers. In proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 2017, pp. 598-605. ISSN: 2331-9852, ISBN: 978-1-5386-2682-5, DOI: 10.1109/IROS.2017.8202213.

In Chapter 7

Authors	Lingni Ma[1]	lingni.ma@in.tum.de
	Jörg Stückler[2]	joerg.stueckler@tuebingen.mpg.de
	Christian Kerl[1]	kerl@in.tum.de
	Daniel Cremers[1]	cremers@in.tum.de
	[1] Technische Universität München	
	[2] RWTH Aachen University	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	© 2017 IEEE	Reprinted with permission from IEEE

Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform. Lingni Ma, Jörg Stückler, Tao Wu and Daniel Cremers. In Arxiv preprint: ARXIV:1808.01834 [cs.CV], 2018.

In Chapter 8

Authors	Lingni Ma[1]	lingni.ma@in.tum.de
	Jörg Stückler[2]	joerg.stueckler@tuebingen.mpg.de
	Tao Wu[1]	tao.wu@in.tum.de
	Daniel Cremers[1]	cremers@in.tum.de
	[1] Technische Universität München	
	[2] Max Planck Institute for Intelligent Systems	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	arXiv.org with a perpetual, non-exclusive license	

Human-in-the-loop Annotation for Large-scale 3D Semantic Datasets. Lingni Ma, Julian Straub, Yufan Chen, Thomas Whelan, Rohan Chabra, Richard A. Newcombe and Daniel Cremers. submitted to ACM SIGGRAPH ASIA 2018.
In Chapter 9

Authors	Lingni Ma[1, 2]	lingni.ma@in.tum.de
	Julian Straub[2]	straub@oculus.com
	Yufan Chen[2]	steven.chen2@oculus.com
	Thomas Whelan[2]	twhelan@oculus.com
	Rohan Chabra[2]	rohanc@cs.unc.edu
	Richard A. Newcombe[2]	newcombe@oculus.com
	Daniel Cremers[1]	cremers@in.tum.de
	[1] Technische Universität München	
	[2] Facebook Reality Lab	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	Submitted to SIGGRAPH ASIA 2018	

Publications Contributed to Supplementary Materials (Part III)

3D Colored Model Generation Based on Multiview Textures and Triangular Mesh. Lingni Ma, Luat Do, Egor Bondarev and Peter H. N. de With. In proc. of Seventh International Conference on Distributed Smart Cameras (ICDSC), Palm Springs, 2013, pp. 1-6. ISBN: 978-1-4799-2164-5, DOI: 10.1109/ICDSC.2013.6778206.

In Chapter 10

Authors	Lingni Ma[1]	lingni.ma@in.tum.de
	Luat Do[1]	luat.do@tue.nl
	Egor Bondarev[1]	e.bondarev@tue.nl
	Peter H. N. de With[1]	p.h.h.de.with@tue.nl
	[1] Technische Universiteit Eindhoven	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	© 2013 IEEE	

Multi-volume Mapping and Tracking for Real-time RGB-D Sensing. Lingni Ma, Egor Bondarev and Peter H. N. de With. In SPIE Image Processing: Algorithms and Systems XIII, San Francisco, 2015, vol. 9399, pp. 1-8. DOI: 10.1117/12.2083350.

In Chapter 10

Authors	Lingni Ma[1]	lingni.ma@in.tum.de
	Egor Bondarev[1]	e.bondarev@tue.nl
	Peter H. N. de With[1]	p.h.h.de.with@tue.nl
	[1] Technische Universiteit Eindhoven	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	© 2015 Society of Photo-Optical Instrumentation Engineers(SPIE)	

Plane Segmentation and Decimation of Point Clouds for 3D Environment Reconstruction. Lingni Ma, Raphale Favier, Luat Do, Egor Bondarev and Peter H. N. de With. In proc. of IEEE 10th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 2013, pp. 43-49. ISSN: 2331-9852, DOI: 10.1109/CCNC.2013.6488423.

In Chapter 11

Authors	Lingni Ma[1]	lingni.ma@in.tum.de
	Raphale Favier[1]	raphael.favier@tue.nl
	Luat Do[1]	luat.do@tue.nl
	Egor Bondarev[1]	e.bondarev@tue.nl
	Peter H. N. de With[1]	p.h.h.de.with@tue.nl
	[1] Technische Universiteit Eindhoven	
Contributions	problem definition	significantly contributed
	literature survey	significantly contributed
	algorithm development	significantly contributed
	method implementation	significantly contributed
	experimental evaluation	significantly contributed
	manuscript preparation	significantly contributed
Copyright	© 2013 IEEE	

Incremental and Batch Planar Simplification of Dense Point Cloud Maps. Thomas Whelan, Lingni Ma, Egor Bondarev, Peter H. N. de With and John McDonald. In Robotics Autonomous Systems. volume 69, issue C, (July 2015), pp. 3-4. ISSN: 0921-8890, DOI: 10.1016/j.robot.2014.08.019.

In Chapter 11

Authors	Thomas Whelan[1]	twhelan@oculus.com
	Lingni Ma[2]	lingni.ma@in.tum.de
	Egor Bondarev[2]	e.bondarev@tue.nl
	Peter H. N. de With[2]	p.h.h.de.with@tue.nl
	John McDonald[1]	johnmcd@cs.nuim.ie
	[1] National University of Ireland Maynooth	
	[2] Technische Universiteit Eindhoven	
Contributions	problem definition	contributed
	literature survey	significantly contributed
	algorithm development	contributed
	method implementation	helped
	experimental evaluation	contributed
	manuscript preparation	contributed
Copyright	© 2014 Elsevier B.V.	

Zusammenfassung

DICHTES, visuelles SLAM (Simultaneous Localization and Mapping) ist ein fundamentales Problem in Computer Vision und Robotik. Viele existierende Algorithmen basieren auf direkten Methoden und primitiven Landmarken während komplexere, globale Zusammenhänge oft ignoriert werden. Diese Dissertation schließt die Lücke zwischen verschiedenen RGB-D Ansätzen - inklusive dichter 3D Rekonstruktion, visueller Lokalisierung und maschinellem Lernen. Der Fokus liegt 1) auf Algorithmen, die komplexe Zusammenhänge und Semantik aus RGB-D Daten extrahieren und 2) auf Methoden, die diese Informationen zur Optimierung von flächendeckender Lokalisierung, Bewegungsabschätzung und aussagekräftigen Szenendarstellung benutzen.

Ein bedeutender Beitrag dieser Arbeit ist die Benutzung von planaren Prioren für SLAM. Planare Oberflächen sind häufig in Innenräumen zu finden und gehören zu semantisch wichtigen Objekten, zum Beispiel Wänden, Böden oder Fenstern. Diese Dissertation entwickelt Methoden um planare Oberflächen in verschiedenartigen Quellen zu finden und Redundanz in dichten Innenraumrekonstruktionen zu verringern, was zu kompakteren Repräsentationen führt allerdings ohne geometrische oder visuelle Informationen zu verlieren. Zusätzlich wird ein kompletter Echtzeit-SLAM Algorithmus - CPA SLAM - vorgestellt, der Innenräume mit globalen, planaren Karten ausstattet, die die Kamerabewegung mit einer Mischung aus Schlüsselbildern und Model basiertem Tracking approximiert und globale Übereinstimmung von allen Daten forciert.

Zusätzlich zu planaren Prioren stellt die Arbeit mehrere Algorithmen vor, die semantische Informationen aus RGB-D Daten mit Hilfe von convolutional Neural Networks (CNNs) extrahieren. FuseNet ist eine neue Netzwerkarchitektur, die beweisbar effizient im Lernen von Eigenschaften basierend auf einzelnen RGB-D Bildern ist. Zusätzlich wurde ein Algorithmus entwickelt, der CNNs reguliert, wenn Ansichten von mehreren Seiten zur Verfügung stehen, was die Geometrie und konsistente Semantiken optimiert. Ein weiterer Beitrag ist die Einführung von diskreten Wavelet Transformationen (DWT) in Encoder-Decoder CNNs. Durch die Benutzung von DWTs und inversen DWTs können Pooling- und Unpoolingoperationen ersetzt werden, was die Detailgenauigkeit der semantischen Segmentierung verbessert.

Der letzte Beitrag ist ein Framework zur semantischen Datenannotation mit menschlicher Interaktion für die Beschriftung. 3D Oberflächen und 2D Videos können mit diesem Algorithmus konsistent und effizient segmentiert und beschriftet werden. Dabei lernt ein Algorithmus diese Segmentierungen für Oberflächen automatisch aus 2D Segmentierungen und Annotationen zu erstellen. Die Arbeit zeigt, dass diese Folgerichtigkeit die Annotation von großen Datenmengen, die für Deep Learning nötig sind, erheblich vereinfacht.

Samenvatting

HET visueel inmeten van een scene met hoge dichtheid is een fundamenteel probleem in computervisie en robotica. Veel bestaande algoritmen hebben directe beeldmethoden en functies op laag niveau verkend, maar hoe ze op hoog niveau kunnen worden geïnterpreteerd en geaggregeerd is aanzienlijk minder onderzocht. Dit proefschrift heeft als doel deze kloof voor RGB-D-visie te overbruggen vanuit verschillende perspectieven, waaronder compacte 3D-reconstructie, visuele simultane lokalisatie en mapping (SLAM) en machine learning. Daarom focust dit proefschrift zich op de volgende punten: i) algoritmen die abstracties op hoog niveau en semantiek uit RGB-D-data halen en ii) methoden die dergelijke informatie gebruiken om dichte mapping te optimaliseren, bewegingsschatting te verbeteren en betekenisvolle scene-representaties verkrijgen.

Een belangrijke bijdrage van dit proefschrift is het gebruik van a priori vlakmodellen voor het visueel inmeten van een scene. In binnenscènes komen vlakke opervlakken vaak voor en zeggen ze iets over de semantiek van de scene, zoals muren, vloeren, ramen en dergelijke. In dit proefschrift worden methoden geïntroduceerd om vlakke oppervlakken in data van verschillende gegevensbronnen te detecteren. Met behulp van a priori vlakmodellen hebben we algoritmen ontwikkeld om redundantie te verminderen in de representatie bij het maken van scans binnenshuis en een compacte reconstructie te verkrijgen zonder geometrische en visuele informatie te verliezen. Verder hebben we een volledig real-time RGB-D SLAM-algoritme ontwikkeld, bekend als CPA SLAM, dat (i) de scènes binnenshuis met globale vlakken modelleert, (ii) camerabeweging schat met een combinatie van kernframe-gebaseerde en modelgebaseerde volgmethoden en (iii) de globale consistentie optimaliseert door middel van vlakmodellen.

Naast de a priori vlakmodellen, hebben we verschillende algoritmen bijgedragen om de semantiek uit de RGB-D-visie te halen met behulp van diepe convolutionele neurale netwerken (CNN's). Eerst introduceren we FuseNet, een nieuwe netwerkarchitectuur die efficiënt is in het leren van functies van RGB-D-beelden met enkel beeld. Verder introduceren we algoritmes om CNN's te regulariseren met multi-viewbeperkingen. Dit doen we op basis van geometrie uit RGB-D-reeksen en door aan te tonen dat een semantische afbeelding kan worden verkregen met een betere consistentie. Een andere bijdrage die in dit proefschrift beschereven wordt is de introductie van discrete wavelet-transformatie (DWT) in zogenaamde encorder-decoder CNN's. Met behulp van DWT en inverse DWT om de pooling- en unpool-bewerkingen respectievelijk te vervangen, hebben we het detailniveau in dichte semantische segmentatie verbeterd. De laatste bijdrage is een semantisch annotatiealgoritme, waarbij de waarheidslabels met behulp van menselijke interactie verkregen worden. Er wordt een op segmentatie gebaseerde vrije-vorm-maas-labeling voorgesteld, waarbij de scene door de mens wordt gecompleteerd, om zo consistente 3D-gelabelde mazen en 2D geannoteerde video's te produceren. Daarnaast hebben we algoritmen ontwikkeld om bijvoorbeeld segmentatie voor 3D-mazen te verkrijgen door 2D-segmentatie van geannoteerde gegevens te leren en samen te voegen. Dit gesloten-lus-label methode is aantoonbaar voordelig voor grootschalige waarheidlabel generatie.

Summary

Semantic Mapping and Tracking with RGB-D Cameras

FOR the past half a century, computer scientists, machine learning experts and mathematicians have been researching about the problem - *"let computer describe what it sees"*. Within the big question, consider the scenarios where autonomous devices need to navigate around the environment and interact with the surroundings. Three problems are fundamental: how to map the environment, how to localize itself onto the map, and what decisions to make. With decades of research, people have been developing direct image methods and exploiting low-level features to model the physical mechanics of the scene. Less explored is how to infer and aggregate high-level abstracts and semantic understandings to assist mapping, tracking and ultimately decision making. In recent years, the increasing availability of RGB-D cameras has inspired many novel solutions to motion estimation, dense reconstruction and visual understandings. In this thesis, we explore algorithms that extract and utilize semantic understandings to optimize dense mapping, improve motion estimation and obtain meaningful scene representations.

Towards the goal of semantic mapping and tracking, we first explore simple geometric primitives, and introduce plane priors for real-time dense mapping and RGB-D SLAM. Planar surfaces are common features of indoor scenes, which often encodes semantics, *e.g.*, walls, floors and ceilings. In Chapter 4 (*Planar Simplification and Texturing of Dense Point Cloud Maps*), we focus on reconstruction of dense point clouds and propose a hybrid mapping algorithm to process planar and non-planar surfaces separately. To this end, we develop a plane segmentation algorithm in Chapter 11 (*RGB-D Mapping with Plane Priors*), which performs more efficiently and yields better sensible segments in comparison to the classic RANSAC-based detection. Further, we introduce a quadtree-based decimation method to reduce approximately 90% of the redundant input planar points. The proposed decimation algorithm enables two triangulation techniques where both produce compact mesh reconstructions for planar surfaces. Our method yields triangulation with no more than 10% of the amount of triangles required without decimation. To transfer the visual context embedded in the dense input to the sparse output, a computationally inexpensive algorithm is developed to generate textures for simplified planar surfaces. We demonstrate our algorithm enhances processing efficiency in particular for large-scale indoor datasets with a parallel multi-threaded implementation. The output models are compact, geometrically accurate and visually appealing, which are also in a useful format for many robotic applications. This algorithm is further extended in Chapter 11 to incrementally process gradually

expanding point cloud maps.

Beyond dense reconstruction, Chapter 5 (*CPA-SLAM: Consistent Plane-Model Alignment for Direct RGB-D SLAM*) integrates plane priors into a real-time RGB-D SLAM algorithm, and proposes the CPA-SLAM algorithm. With CPA-SLAM, instead of building a dense map from all available RGB-D data, we represent the scene model as global planar map. To this end, the individual planes are detected from the depth image of keyframes, properly associated across all keyframes and optimized in a global graph optimization. Based on the planar map representation, a novel tracking method is proposed that combines direct image alignment and global model alignment into a probabilistic formation and estimates the motion with expectation-maximization. This tracking formulation reduces per-frame tracking drift and establishes additional constraints among non-overlapping keyframes once a common plane is observed. The keyframe poses and the plane model are optimized concurrently in one graph. Our method exhibits state-of-the-art accuracy on publicly available benchmark datasets and is capable of real-time performance.

Moving towards semantics, this thesis also investigates semantic scene understanding with deep CNNs. As a fundamental work to learn from RGB-D data, Chapter 6 (*FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture*) proposes a novel fusion-based CNN network, namely FuseNet. FuseNet exploits two network branches to filter the color and the depth images separately, while consistently combines the depth features into the color branch as the network goes deeper. Shown in experiments, FuseNet outperforms the existing CNN-based networks on the challenging RGB-D semantic segmentation benchmark. Following the work on FuseNet for single-view learning, Chapter 7 (*Multi-View Deep Learning for Consistent Semantic Mapping with RGB-D Cameras*) extends it to learn consistent semantic mapping from multi-view RGB-D sequences. Using the camera trajectories obtained by RGB-D SLAM algorithm, our method warps network outputs of multiple viewpoints into a common reference view (in correspondence with the keyframe in SLAM settings) at different stages to enforce invariant feature learning under viewpoint changes. We demonstrate the superior performance of multi-view consistency training and Bayesian fusion on the NYUDv2 13-class and 40-class semantic segmentation benchmark, which sets a new state-of-the-art performance using an end-to-end trained network for single-view predictions as well as multi-view fused semantic segmentation without further post-processing. In Chapter 12 (*Semantic RGB-D Mapping*), we demonstrate semantic RGB-D mapping by fusing multi-view segmentations from RGB-D sequences in a probabilistic formation.

In Chapter 8 (*Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform*), we explore CNN architectures to learn detailed semantic segmentation, or in a broader perspective dense pixelwise predictions. Motivated by the structural analogy between multi-resolution wavelet analysis and the pooling/unpooling operations in CNNs, we introduce discrete wavelet transform (DWT) into the encoder-decoder CNNs and develop wavelet-based networks, namely WCNN. In the proposed network, pooling in the encoder and unpooling in the decoder are replaced by DWT and inverse DWT, respectively. The low-frequency wavelet coefficients considered as the output of pooling, are processed with further convolutional layers. The high-frequency wavelet coefficients are cached during the encoder stage, and later combined with the corresponding coarse-resolution feature map to compose inverse DWT at decoder with skip connections. We show that without additional training parameters over the baseline CNNs, WCNNs systematically improve fine-resolution predictions.

Consider the importance of high-quality large-scale RGB-D semantic ground-truth datasets, Chapter 9 (*Human-in-the-loop Annotation for Large-scale 3D Semantic Datasets*) proposes a close-loop annotation algorithm. At the annotation stage, we develop a unified semantic instance labeling tool to perform segmentation-based free-form annotation on meshes, and additionally inpaint missing surface reconstructions with human aid. We further rendered the labeled 3D mesh into RGB-D sequences to obtain fully annotated video and optimize the results with joint bilateral filtering. As shown in experiments, the proposed method produces detailed annotations that are consistent between 3D meshes and 2D images. A human-in-the-loop labeling scheme is further developed to alleviate human efforts for large-scale dataset collection. To this end, the existing annotations are used to train CNNs for 2D instance segmentation and the predictions are fused back to 3D meshes based on geometric cues. Evaluated on an indoor dataset of 20 rooms, the proposed approach generate almost correct semantic segmentation for human to refine over a large office space.

Own Publications

[1] **E. Bondarev, F. Heredia, R. Favier, L. Ma, and P. H. N. de With**. On Photo-realistic 3D Reconstruction of Large-scale and Arbitrary-shaped Environments. In: *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. Jan. 2013, pp. 621–624.

[2] **M. Dzitsiuk, J. Sturm, R. Maier, L. Ma, and D. Cremers**. De-noising, Stabilizing and Completing 3D Reconstructions On-the-go Using Plane Priors. In: *proc. of International Conference on Robotics and Automation (ICRA)*. Singapore, July 2017, pp. 3976–3983. ISBN: 978-1-5090-4634-8 (page 141).

[3] **C. Hazirbas, L. Ma, C. Domokos, and D. Cremers**. FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture. In: *proc. of Asian Conference on Computer Vision (ACCV)*. Cham: Springer International Publishing, Nov. 2017, pp. 213–228. ISBN: 978-3-319-54181-5 (pages 13, 16, 18, 104–106, 108, 112, 113, 115, 116, 119, 183, 184).

[4] **L. Ma, L. Do, E. Bondarev, and P. H. N. de With**. 3D Colored Model Generation Based on Multiview Textures and Triangular Mesh. In: *2013 Seventh International Conference on Distributed Smart Cameras (ICDSC)*. Oct. 2013, pp. 1–6. ISBN: 978-1-4799-2164-5 (pages 13, 15, 19).

[5] **L. Ma, C. Kerl, J. Stückler, and D. Cremers**. CPA-SLAM: Consistent Plane-model Alignment for Direct RGB-D SLAM. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm, Sweden, May 2016, pp. 1285–1291 (pages 13, 16, 18).

[6] **L. Ma, J. Stückler, C. Kerl, and D. Cremers**. Multi-view deep learning for consistent semantic mapping with RGB-D cameras. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 598–605 (pages 13, 17, 18).

[7] **L. Ma, T. Whelan, E. Bondarev, P. H. N. de With, and J. McDonald**. Planar Simplification and Texturing of Dense Point Cloud Maps. In: *2013 European Conference on Mobile Robots (ECMR)*. Sept. 2013, pp. 164–171. ISBN: 978-1-4799-0263-7 (pages 13, 15, 18).

[8] **L. Ma, E. Bondarev, and P. H. N. de With**. Multi-volume Mapping and Tracking for Real-time RGB-D Sensing. In: *SPIE Image Processing: Algorithms and Systems XIII* 9399 (2015), pp. 1–8 (pages 13, 15, 19).

[9] **L. Ma, R. Favier, L. Do, E. Bondarev, and P. H. N. de With**. Plane Segmentation and Decimation of Point Clouds for 3D Environment Reconstruction. In: *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. Jan. 2013, pp. 43–49 (pages 13, 15, 19, 57–59, 61).

[10] **L. Ma, J. Straub, Y. Chen, T. Whelan, R. Chabra, R. Newcombe, and D. Cremers**. Human-in-the-loop Annotation for Large-scale 3D Semantic Datasets. In: *SIGGRAPH Asia (submitted)*. 2018 (pages 13, 18, 19).

[11] **L. Ma, J. Stückler, T. Wu, and D. Cremers**. Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform. In: *Arxiv preprint* (2018). arXiv: 1808.01834 [cs.CV] (pages 13, 17, 18).

[12] **T. Whelan, L. Ma, E. Bondarev, P. H. N. de With, and J. McDonald**. Incremental and Batch Planar Simplification of Dense Point Cloud Maps. In: *Robotics Autonomous Systems (RAS)* 69.C (July 2015), pp. 3–14. ISSN: 0921-8890 (pages 13, 16, 19).

Bibliography

[13] **M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng**. TensorFlow: A System for Large-scale Machine Learning. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 978-1-931971-33-1 (pages 50, 131).

[14] **Y. Alshawabkeh and N. Haala**. Automatic multi-image photo texturing of complex 3D scenes. eng. In: *CIPA 2005 International Symposium, Vol. 34-5/C34, pp. 68-73*. Universitt Stuttgart, 2005 (page 163).

[15] **I. Armeni, A. Sax, A. R. Zamir, and S. Savarese**. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. In: *ArXiv e-prints* (Feb. 2017). arXiv: 1702.01105 (pages 12, 141).

[16] **I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese**. 3D Semantic Parsing of Large-Scale Indoor Spaces. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pages 5, 107).

[17] **V. Badrinarayanan, A. Kendall, and R. Cipolla**. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2017) (pages 10, 50, 89, 92, 122).

[18] **S. Baker and I. Matthews**. Lucas-Kanade 20 Years On: A Unifying Framework. In: *International Journal of Computer Vision (IJCV)* 56.3 (Feb. 2004), pp. 221–255. ISSN: 0920-5691 (page 37).

[19] **Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle**. Greedy Layer-Wise Training of Deep Networks. In: *proc. of Advances in Neural Information Processing Systems (NIPS)*. 2007 (pages 10, 105).

[20] **J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-farley, and Y. Bengio**. Theano: A CPU and GPU math compiler in python. In: *Proceedings of the 9th Python in Science Conference*. 2010, pp. 3–10 (page 50).

[21] **F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin**. The Ball-Pivoting Algorithm for Surface Reconstruction. In: *IEEE Trans. on Visualization and Computer Graphics* 5.4 (1999), pp. 349–359. ISSN: 1077-2626 (pages 5, 56).

[22] **P. J. Besl and N. D. McKay**. A Method for Registration of 3-D Shapes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 14.2 (Feb. 1992), pp. 239–256 (pages 7, 37, 76, 181).

[23] **C. M. Bishop**. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738 (pages 23, 42, 78).

[24] **L. Bottou**. Stochastic gradient descent tricks. In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 421–436 (pages 27, 96, 113).

[25] **M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst**. Geometric Deep Learning: Going beyond Euclidean data. In: *IEEE Signal Processing Magazine* 34.4 (July 2017), pp. 18–42 (page 193).

[26] **W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki**. Scene Labeling With LSTM Recurrent Neural Networks. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 3547–3555 (pages 88, 90).

[27] **E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers**. Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions. In: *Robotics: Science and Systems Conference (RSS)*. June 2013 (page 8).

[28] **A. Canziani, A. Paszke, and E. Culurciello**. An analysis of deep neural network models for practical applications. In: *arXiv preprint arXiv:1605.07678* (2016) (page 50).

[29] **S. Chandra and I. Kokkinos**. Fast, Exact and Multi-scale Inference for Semantic Image Segmentation with Deep Gaussian CRFs. In: *proc. of European Conference on Computer Vision (ECCV)*. 2016, pp. 402–418 (page 124).

[30] **A. Chang, A. Dai, T. Funkhouser, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang**. Matterport3D: Learning from RGB-D Data in Indoor Environments. In: *proc. of International Conference on 3D Vision (3DV)*. 2017 (pages 12, 141, 193).

[31] **L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille**. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 40 (2018), pp. 834–848 (pages 10, 50, 122, 123, 129).

[32] **L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille**. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In: *proc. of International Conference on Learning Representations (ICLR)*. San Diego, CA, USA, May 2015 (pages 10, 44, 50, 88, 89, 122, 123, 129).

[33] **Y. Chen and G. Medioni**. Object modeling by registration of multiple range images. In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. Apr. 1991, 2724–2729 vol.3 (page 37).

[34] **D.-A. Clevert, T. Unterthiner, and S. Hochreiter**. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In: *arXiv preprint* abs/1511.07289 (2015) (page 46).

[35] **R. Collobert, S. Bengio, and J. Mariéthoz**. *Torch: a modular machine learning software library*. Tech. rep. Idiap, 2002 (page 50).

[36] **M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele**. The Cityscapes Dataset for Semantic Urban Scene Understanding. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3213–3223 (pages 17, 130).

[37] **C. Couprie, C. Farabet, L. Najman, and Y. LeCun**. Indoor Semantic Segmentation using Depth Information. In: *proc. of International Conference on Learning Representations (ICLR)*. Apr. 2013 (pages 10, 88, 105, 112).

[38] **B. Curless and M. Levoy**. A Volumetric Method for Building Complex Models from Range Images. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 303–312. ISBN: 0-89791-746-4 (pages 5, 7, 8, 39, 171).

[39] **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 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pages 12, 137, 140–142, 153, 193).

[40] **A. Dai, M. Niessner, M. Zollöfer, S. Izadi, and C. Theobalt**. BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration. In: *ACM Transactions on Graphics* (2017) (pages 7, 8, 12, 141).

[41] **A. Dai, C. R. Qi, and M. Niessner**. Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pages 141, 142, 193).

[42] **F. Darema, D. George, V. Norton, and G. Pfister**. A single-program-multiple-data computational model for EPEX/FORTRAN. In: *Parallel Computing* 7.1 (1988), pp. 11–24. ISSN: 0167-8191 (page 58).

[43] **V. Domiter and B. Zalik**. Sweep-line algorithm for constrained Delaunay triangulation. In: *Int. J. Geogr. Inf. Sci.* 22.4 (Jan. 2008), pp. 449–462. ISSN: 1365-8816 (pages 5, 60).

[44] **A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox**. FlowNet: Learning Optical Flow With Convolutional Networks. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015 (pages 109, 123).

[45] **A. Dosovitskiy, J. Tobias Springenberg, and T. Brox**. Learning to generate chairs with convolutional neural networks. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1538–1546 (pages 10, 123, 129).

[46] **M. Dou, L. Guan, J.-M. Frahm, and H. Fuchs**. Exploring High-Level Plane Primitives for Indoor 3D Reconstruction with a Hand-held RGB-D Camera. In: *Computer Vision - ACCV Workshops*. 2012, pp. 94–108 (pages 9, 73).

[47] **J. Duchi, E. Hazan, and Y. Singer**. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159. ISSN: 1532-4435 (page 27).

[48] **M. Duckham, L. Kulik, M. Worboys, and A. Galton**. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. In: *Pattern Recogn.* 41.10 (2008), pp. 3224–3236. ISSN: 0031-3203 (page 59).

[49] **D. Eigen and R. Fergus**. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. 2015 (pages 10, 96, 105, 112, 114–116, 119).

[50] **F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard**. An Evaluation of the RGB-D SLAM System. In: *proc. of International Conference on Robotics and Automation (ICRA)*. May 2012 (pages 83, 84).

[51] **J. Engel, V. Koltun, and D. Cremers**. Direct Sparse Odometry. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* PP.99 (2017), pp. 1–1. ISSN: 0162-8828 (page 7).

[52] **J. Engel, T. Schöps, and D. Cremers**. LSD-SLAM: Large-Scale Direct Monocular SLAM. In: *proc. of European Conference on Computer Vision (ECCV)*. 2014 (page 7).

[53] **M. F. Fallon, H. Johannsson, and J. J. Leonard**. Efficient Scene Simulation for Robust Monte Carlo Localization using an RGB-D Camera. In: *proc. of International Conference on Robotics and Automation (ICRA)*. St Paul, Minnesota, USA, May 2012 (pages 56, 66).

[54] **C. Farabet, C. Couprie, L. Najman, and Y. LeCun**. Learning Hierarchical Features for Scene Labeling. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 35.8 (Aug. 2013), pp. 1915–1929. ISSN: 0162-8828 (page 123).

[55] **P. F. Felzenszwalb and D. P. Huttenlocher**. Efficient Graph-Based Image Segmentation. In: *Int. J. Comput. Vision* 59.2 (Sept. 2004), pp. 167–181. ISSN: 0920-5691 (page 144).

[56] **C. Feng, Y. Taguchi, and V. R. Kamat**. Fast Plane Extraction in Organized Point Clouds Using Agglomerative Hierarchical Clustering. In: *proc. of International Conference on Robotics and Automation (ICRA)*. 2014 (page 75).

[57] **N. Fioraio and L. di Stefano**. Joint Detection, Tracking and Mapping by Semantic Bundle Adjustment. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013, pp. 1538–1545 (page 73).

[58] **S. Fortune**. Handbook of Discrete and Computational Geometry. In: ed. by **J. E. Goodman and J. O'Rourke**. Boca Raton, FL, USA: CRC Press, Inc., 1997. Chap. Voronoi Diagrams and Delaunay Triangulations, pp. 377–388. ISBN: 0-8493-8524-5 (page 5).

[59] **Y. Gal and Z. Ghahramani**. Dropout As a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In: *proc. of International Conference on Machine Learning (ICML)*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1050–1059 (page 89).

[60] **A. P. Gee, D. Chekhlov, W. W. Mayol-Cuevas, and A. Calway**. Discovering Planes and Collapsing the State Space in Visual SLAM. In: *proc. of British Machine Vision Conference (BMVC)*. 2007 (page 73).

[61] **G. Ghiasi and C. C. Fowlkes**. Laplacian Pyramid Reconstruction and Refinement for Semantic Segmentation. In: *proc. of European Conference on Computer Vision (ECCV)*. 2016, pp. 519–534 (page 124).

[62] **R. Girshick, J. Donahue, T. Darrell, and J. Malik**. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (pages 10, 105).

[63] **X. Glorot and Y. Bengio**. Understanding the difficulty of training deep feedforward neural networks. In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*. 2010 (page 52).

[64] **M. Gopi and S. Krishnan**. A fast and efficient projection-based approach for surface reconstruction. In: *Proc. Computer Graphics and Image Processing*. 2002, pp. 179–186 (pages 5, 57).

[65] **L. Grammatikopoulos, I. Kalisperakis, G. Karras, and E. Petsa**. Automatic multi-view texture mapping of 3D surface projections. In: *3D Virtual Reconstruction & Visualization of Complex Architectures*. 2007, pp. 12–13 (page 163).

[66] **A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber**. A Novel Connectionist System for Unconstrained Handwriting Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 31.5 (May 2009), pp. 855–868. ISSN: 0162-8828 (page 43).

[67] **G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard**. A tutorial on graph-based SLAM. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43 (page 8).

[68] **S. Gupta, P. Arbelaez, and J. Malik**. Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013 (page 112).

[69] **S. Gupta, R. Girshick, P. Arbeláez, and J. Malik**. Learning rich features from RGB-D images for object detection and segmentation. In: *proc. of European Conference on Computer Vision (ECCV)*. 2014 (pages 10, 88, 90, 91, 105, 112).

[70] **D. Gutierrez-Gomez, W. Mayol-Cuevas, and J. Guerrero**. Dense RGB-D Visual Odometry Using Inverse Depth. In: *Robotics Autonomous Systems (RAS)* 75.PB (Jan. 2016), pp. 571–583. ISSN: 0921-8890 (page 38).

[71] **A. Handa, T. Whelan, J. McDonald, and A. Davison**. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In: *proc. of International Conference on Robotics and Automation (ICRA)*. May 2014 (page 82).

[72] **A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla**. Scenenet: Understanding real world indoor scenes with synthetic data. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pages 112, 114).

[73] **C. Häne, C. Zach, A. Cohen, and M. Pollefeys**. Dense Semantic 3D Reconstruction. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 39.9 (Sept. 2017), pp. 1730–1743. ISSN: 0162-8828 (page 5).

[74] **B. Hariharan, P. A. Arbeláez, R. B. Girshick, and J. Malik**. Hypercolumns for object segmentation and fine-grained localization. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 447–456 (page 123).

[75] **K. He, X. Zhang, S. Ren, and J. Sun**. Deep Residual Learning for Image Recognition. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 770–778 (pages 9, 10, 27, 44, 47, 49, 51, 122–124, 126, 127, 131, 132).

[76] **K. He, G. Gkioxari, P. Dollár, and R. B. Girshick**. Mask R-CNN. In: *proc. of IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2980–2988 (pages 11, 138, 139, 141).

[77] **K. He, X. Zhang, S. Ren, and J. Sun**. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. 2015 (pages 46, 52, 113, 131).

[78] **K. He, X. Zhang, S. Ren, and J. Sun**. Identity mappings in deep residual networks. In: *proc. of European Conference on Computer Vision (ECCV)*. Springer. 2016, pp. 630–645 (pages 10, 49).

[79] **Y. He, W. Chiu, M. Keuper, and M. Fritz**. STD2P: RGBD Semantic Segmentation Using Spatio-Temporal Data Driven Pooling. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (page 107).

[80] **P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox**. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. In: *The Int. Journal of Robotics Research* (2012) (pages 8, 56).

[81] **A. Hermans, G. Floros, and B. Leibe**. Dense 3D Semantic Mapping of Indoor Scenes from RGB-D Images. In: *proc. of International Conference on Robotics and Automation (ICRA)*. 2014 (pages 5, 107, 112, 114, 184, 186).

[82] **S. Hochreiter and J. Schmidhuber**. Long Short-term Memory. In: 9 (Dec. 1997), pp. 1735–80 (page 43).

[83] **B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung**. SceneNN: A Scene Meshes Dataset with aNNotations. In: *proc. of International Conference on 3D Vision (3DV)*. 2016 (pages 12, 141, 193).

[84] **H. Huang, S. Wu, D. Cohen-Or, M. Gong, H. Zhang, G. Li, and B. Chen**. L1-medial Skeleton of Point Cloud. In: *ACM Transactions on Graphics* 32.4 (July 2013), 65:1–65:8. ISSN: 0730-0301 (page 147).

[85] **J. Huang, A. Dai, L. Guibas, and M. Niessner**. 3DLite: Towards Commodity 3D Scanning for Content Creation. In: *ACM Transactions on Graphics* (2017) (page 141).

[86] **P. J. Huber**. The 1972 Wald Lecture Robust Statistics: A Review. In: *The Annals of Mathematical Statistics* 43.4 (1972), pp. 1041–1067. ISSN: 00034851 (page 31).

[87] **S. Ioffe and C. Szegedy**. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *proc. of International Conference on Machine Learning (ICML)*. Vol. 37. JMLR Proceedings. JMLR.org, 2015, pp. 448–456 (pages 47, 93).

[88] **M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu koray**. Spatial Transformer Networks. In: *proc. of Advances in Neural Information Processing Systems (NIPS)*. 2015 (page 110).

[89] **A. C. Jalba and J. B. T. M. Roerdink**. Efficient surface reconstruction from noisy data using regularized membrane potentials. In: *IEEE Transactions on Image Processing* 18.5 (May 2009), pp. 1119–1134. ISSN: 1057-7149 (page 56).

[90] **A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell**. A category-level 3-D object dataset: Putting the Kinect to work. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. Nov. 2011, pp. 1168–1174 (page 12).

[91] **Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell**. Caffe: Convolutional Architecture for Fast Feature Embedding. In: *Proceedings of the 22Nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: ACM, 2014, pp. 675–678. ISBN: 978-1-4503-3063-3 (pages 50, 96, 113).

[92] **G. S. Johnson, J. Lee, C. A. Burns, and W. R. Mark**. The irregular Z-buffer: Hardware acceleration for irregular data structures. In: *atg* 24.4 (2005), pp. 1462–1482.

[93] **M. Kaess, A. Ranganathan, and F. Dellaert**. iSAM: Incremental Smoothing and Mapping. In: *IEEE Transactions on Robotics* 24.6 (Dec. 2008), pp. 1365–1378. ISSN: 1552-3098 (page 8).

[94] **M. Kazhdan, M. Bolitho, and H. Hoppe**. Poisson surface reconstruction. In: *Proc. of the 4th Eurographics Symposium on Geometry Processing*. 2006, pp. 61–70. ISBN: 3-905673-36-3 (pages 5, 56).

[95] **M. Kazhdan and H. Hoppe**. Screened Poisson Surface Reconstruction. In: *ACM Transactions on Graphics* 32.3 (July 2013), 29:1–29:13. ISSN: 0730-0301 (pages 5, 12).

[96] **M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb**. Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion. In: *proc. of International Conference on 3D Vision (3DV)*. 2013 (pages 9, 73).

[97] **A. Kendall, V. Badrinarayanan, and R. Cipolla**. Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding. In: *arXiv preprint* (2015) (pages 89, 92, 93, 98).

[98] **C. Kerl, J. Sturm, and D. Cremers**. Dense Visual SLAM for RGB-D Cameras. In: *proc. of International Conference on Intelligent Robots and Systems (IROS)*. 2013 (pages 7, 8, 38, 72, 76, 80, 83, 84, 110, 112).

[99] **C. Kerl, J. Sturm, and D. Cremers**. Robust Odometry Estimation for RGB-D Cameras. In: *proc. of International Conference on Robotics and Automation (ICRA)*. May 2013 (page 37).

[100] **K. Khoshelham**. Accuracy analysis of kinect depth data. English. In: *ISPRS workshop laser scanning 2011, Calgary, Canada, 29-31 August 2011*. Ed. by **D. Lichti and A. Habib**. International Society for Photogrammetry and Remote Sensing (ISPRS), Aug. 2011, pp. – (page 32).

[101] **D. P. Kingma and J. Ba**. Adam: A Method for Stochastic Optimization. In: 2015 (pages 27, 28).

[102] **P. Krähenbühl and V. Koltun**. Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials. In: *proc. of Advances in Neural Information Processing Systems (NIPS)*. 2011 (pages 50, 114, 122, 139, 184–186).

[103] **P. Krähenbühl and V. Koltun**. Parameter Learning and Convergent Inference for Dense Random Fields. In: *proc. of International Conference on Machine Learning (ICML)*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, pp. III-513–III-521 (page 122).

[104] **A. Krizhevsky, I. Sutskever, and G. E. Hinton**. ImageNet Classification with Deep Convolutional Neural Networks. In: *proc. of Advances in Neural Information Processing Systems (NIPS)*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105 (pages 9, 27, 45, 49, 121, 123).

[105] **R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard**. G2O: A General Framework for Graph Optimization. In: *proc. of International Conference on Robotics and Automation (ICRA)*. May 2011 (page 82).

[106] **A. Kundu, V. Vineet, and V. Koltun**. Feature Space Optimization for Semantic Video Segmentation. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (page 107).

[107] **I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab**. Deeper Depth Prediction with Fully Convolutional Residual Networks. In: *proc. of International Conference on 3D Vision (3DV)* (2016), pp. 239–248 (pages 10, 50, 121, 123).

[108] **Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel**. Backpropagation Applied to Handwritten Zip Code Recognition. In: *Neural Comput.* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667 (page 42).

[109] **Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner**. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219 (page 49).

[110] **C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu**. Deeply-Supervised Nets. In: *Proc. of the 18th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*. 2015 (page 109).

[111] **H. P. A. Lensch, W. Heidrich, and H.-P. Seidel**. A Silhouette-Based Algorithm for Texture Registration and Stitching. In: *Graphical Models* 63.4 (Dec. 18, 2003), pp. 245–262 (page 163).

[112] **G. Li, L. Liu, H. Zheng, and N. J. Mitra**. Analysis, Reconstruction and Manipulation Using Arterial Snakes. In: *ACM SIGGRAPH Asia 2010 Papers*. SIGGRAPH ASIA '10. Seoul, South Korea: ACM, 2010, 152:1–152:10. ISBN: 978-1-4503-0439-9 (page 141).

[113] **Z. Li, Y. Gan, X. Liang, Y. Yu, H. Cheng, and L. Lin**. LSTM-CF: Unifying Context Modeling and Fusion with LSTMs for RGB-D Scene Labeling. In: *proc. of European Conference on Computer Vision (ECCV)*. 2016 (page 107).

[114] **Z. Li, Y. Gan, X. Liang, Y. Yu, H. Cheng, and L. Lin**. RGB-D Scene Labeling with Long Short-Term Memorized Fusion Model. In: *arXiv preprint* (2016). arXiv: `1604.05000` (pages 91, 98).

[115] **G. Lin, A. Milan, C. Shen, and I. Reid**. RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pages 10, 50, 107, 121, 123).

[116] **G. Lin, C. Shen, A. van den Hengel, and I. Reid**. Exploring Context with Deep Structured models for Semantic Segmentation. In: *arXiv preprint* (2016) (pages 11, 89, 90, 97, 98, 107, 112).

[117] **G. Lin, C. Shen, A. v. d. Hengel, and I. Reid**. Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pages 11, 88, 89, 107, 122, 124).

[118] **M. Lin, Q. Chen, and S. Yan**. Network in network. In: *arXiv preprint arXiv:1312.4400* (2013) (page 44).

[119] **T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick**. Microsoft COCO: Common Objects in Context. In: *proc. of European Conference on Computer Vision (ECCV)*. Oral. Zurich, Jan. 2014 (page 141).

[120] **F. Liu, C. Shen, and G. Lin**. Deep convolutional neural fields for depth estimation from a single image. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 5162–5170 (page 123).

[121] **F. Liu, C. Shen, G. Lin, and I. Reid**. Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 38.10 (Oct. 2016), pp. 2024–2039. ISSN: 0162-8828.

[122] **Z. Liu, X. Li, P. Luo, C. C. Loy, and X. Tang**. Semantic Image Segmentation via Deep Parsing Network. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1377–1385 (page 124).

[123] **J. Long, E. Shelhamer, and T. Darrell**. Fully Convolutional Networks for Semantic Segmentation. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pages 10, 44, 46, 48, 50, 88, 89, 98, 105, 112, 123, 129).

[124] **W. E. Lorensen and H. E. Cline**. Marching cubes: A high resolution 3D surface construction algorithm. In: *ACM siggraph computer graphics*. Vol. 21. 4. ACM. 1987, pp. 163–169 (pages 5, 39, 147).

[125] **S. Mallat**. *A Wavelet Tour of Signal Processing: The Sparse Way*. 3rd. Academic Press, 2009 (page 125).

[126] **J. Marquegnies**. *Document layout analysis in SCRIBO*. Tech. rep. CSI Seminar 1102. Research and Development Laboratory, EPITA, `http://www.lrde.epita.fr/dload/20110704-Seminar/1102.pdf`, July 2011 (page 61).

[127] **J. Martinez-Carranza and A. Calway**. Unifying Planar and Point Mapping in Monocular SLAM. In: *proc. of British Machine Vision Conference (BMVC)*. 2010 (page 73).

[128] **Z. C. Marton, R. B. Rusu, and M. Beetz**. On fast surface reconstruction methods for large and noisy point clouds. In: *proc. of International Conference on Robotics and Automation (ICRA)*. 2009, pp. 3218–3223 (pages 5, 57).

[129] **J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst**. Geodesic Convolutional Neural Networks on Riemannian Manifolds. In: *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. Dec. 2015, pp. 832–840 (pages 142, 193).

[130] **J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger**. SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks. In: *arXiv preprint* (2016). ArXiv: `1609.05130` (page 107).

[131] **O. Miksik, V. Vineet, M. Lidegaard, R. Prasaath, M. Niessner, S. Golodetz, S. L. Hicks, P. Pérez, S. Izadi, and P. H. Torr**. The Semantic Paintbrush: Interactive 3D Mapping and Recognition in Large Outdoor Spaces. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. Seoul, Republic of Korea: ACM, 2015, pp. 3317–3326 (page 139).

[132] **N. J. Mitra, A. Nguyen, and L. Guibas**. Estimating Surface Normals in Noisy Point Cloud Data. In: *special issue of International Journal of Computational Geometry and Applications*. Vol. 14. 4–5. 2004, pp. 261–276 (page 40).

[133] **F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein**. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2017, pp. 5425–5434 (pages 142, 193).

[134] **R. Mur-Artal, J. M. M. Montiel, and J. D. Tards**. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163. ISSN: 1552-3098 (pages 6, 7).

[135] **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 International Symposium on Mixed and Augmented Reality (ISMAR)*. 2011, pp. 127–136. ISBN: 978-1-4577-2183-0 (pages 7, 15, 37, 39, 80, 139, 141, 161, 167, 171).

[136] **R. A. Newcombe, S. J. Lovegrove, and A. J. Davison**. DTAM: Dense Tracking and Mapping in Real-time. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2320–2327. ISBN: 978-1-4577-1101-5 (page 7).

[137] **D. T. Nguyen, B.-S. Hua, L.-F. Yu, and S.-K. Yeung**. A Robust 3D-2D Interactive Tool for Scene Segmentation and Annotation. In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* (2017) (pages 12, 137, 140, 141, 145, 153).

[138] **M. Niessner, M. Zollhöfer, S. Izadi, and M. Stamminger**. Real-time 3D Reconstruction at Scale Using Voxel Hashing. In: *ACM Transactions on Graphics* 32.6 (Nov. 2013), 169:1–169:11. ISSN: 0730-0301 (pages 5, 8).

[139] **H. Noh, S. Hong, and B. Han**. Learning Deconvolution Network for Semantic Segmentation. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. 2015 (pages 10, 50, 88, 89, 92, 105, 108, 122, 123, 125).

[140] **A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer**. Automatic differentiation in pytorch. In: (2017) (page 50).

[141] **B. Pateiro-López and A. Rodríguez-Casal**. Generalizing the Convex Hull of a Sample: The R Package alphahull. In: *Journal of Statistical Software* 34.i05 (2010) (page 59).

[142] **M. Pauly, M. Gross, and L. P. Kobbelt**. Efficient simplification of point-sampled surfaces. In: *Proc. of the Conference on Visualization*. VIS '02. Boston, Massachusetts: IEEE Computer Society, 2002, pp. 163–170. ISBN: 0-7803-7498-3 (pages 40, 176).

[143] **H. Pfister, M. Zwicker, J. van Baar, and M. Gross**. Surfels: Surface Elements As Rendering Primitives. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 335–342. ISBN: 1-58113-208-5 (page 8).

[144] **P. O. Pinheiro and R. Collobert**. Recurrent Convolutional Neural Networks for Scene Labeling. In: *proc. of International Conference on Machine Learning (ICML)*. Beijing, China, June 2014 (pages 88, 90).

[145] **T. Pohlen, A. Hermans, M. Mathias, and B. Leibe**. Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3309–3318 (pages 10, 50, 121, 123, 131, 136).

[146] **M. Previtali, L. Barazzetti, and M. Scaioni**. An automated and accurate procedure for texture mapping from images. In: *18th International Conference on Virtual Systems and Multimedia*. Sept. 2012, pp. 591–594.

[147] **C. R. Qi, H. Su, K. Mo, and L. J. Guibas**. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017 (pages 11, 142).

[148] **C. R. Qi, H. Su, K. Mo, and L. J. Guibas**. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In: *proc. of Advances in Neural Information Processing Systems (NIPS)*. Dec. 2017 (pages 11, 142).

[149] **T. Rabbani, F. van Den Heuvel, and G. Vosselmann**. Segmentation of point clouds using smoothness constraint. In: *Inter. Archives of Photo. Remote Sensing and Spatial Info. Sciences* 36.5 (2006), pp. 1–6 (pages 175–177).

[150] **J. Redmon and A. Farhadi**. YOLO9000: Better, Faster, Stronger. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 6517–6525 (page 141).

[151] **S. Ren, K. He, R. Girshick, and J. Sun**. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 39.6 (June 2017), pp. 1137–1149. ISSN: 0162-8828 (page 11).

[152] **G. Riegler, A. O. Ulusoy, and A. Geiger**. OctNet: Learning Deep 3D Representations at High Resolutions. In: *arXiv preprint* abs/1611.05009 (2016) (pages 11, 107).

[153] **O. Ronneberger, P.Fischer, and T. Brox**. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. (available on arXiv:1505.04597 [cs.CV]). Springer, 2015, pp. 234–241 (page 48).

[154] **H. Roth and M. Vona**. Moving volume kinectFusion. In: *proc. of British Machine Vision Conference (BMVC)*. BMVC. 2012, pp. 1–11 (page 171).

[155] **E. Rublee, V. Rabaud, K. Konolige, and G. Bradski**. ORB: An efficient alternative to SIFT or SURF. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. IEEE. 2011, pp. 2564–2571 (page 6).

[156] **D. E. Rumelhart, G. E. Hinton, and R. J. Williams**. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. In: ed. by **D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group**. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X (page 42).

[157] **O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei**. ImageNet Large Scale Visual Recognition Challenge. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252 (pages 49, 96, 141).

[158] **B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman**. LabelMe: A Database and Web-Based Tool for Image Annotation. In: *International Journal of Computer Vision (IJCV)* 77.1-3 (May 2008), pp. 157–173. ISSN: 0920-5691 (pages 12, 137).

[159] **M. Sainz and R. Pajarola**. Point-based rendering techniques. In: *Computers and Graphics* 28.6 (Dec. 2004), pp. 869–879. ISSN: 0097-8493.

[160] **R. F. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison**. Dense Planar SLAM. In: *proc. of International Symposium on Mixed and Augmented Reality (ISMAR)*. 2014 (pages 9, 73, 83).

[161] **R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison**. SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 1352–1359 (pages 9, 73, 107, 193).

[162] **C. E. Scheidegger, S. Fleishman, and C. T. Silva**. Triangulating point set surfaces with bounded error. In: *Proc. of the 3rd Eurographics symposium on Geometry Proc.* Vienna, Austria: Eurographics Association, 2005. ISBN: 3-905673-24-X (page 56).

[163] **F. Servant, E. Marchand, P. Houlier, and I. Marchal**. Visual planes-based simultaneous localization and model refinement for augmented reality. In: *proc. of International Conference on Pattern Recognition (ICPR)*. 2008 (page 73).

[164] **N. Silberman, D. Hoiem, P. Kohli, and R. Fergus**. Indoor Segmentation and Support Inference from RGBD Images. In: *proc. of European Conference on Computer Vision (ECCV)*. 2012 (pages 12, 17, 112, 137, 140).

[165] **K. Simonyan and A. Zisserman**. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *proc. of International Conference on Learning Representations (ICLR)* (2015) (pages 9, 10, 49, 51, 89, 93, 96, 113, 122, 124).

[166] **S. Song, S. P. Lichtenberg, and J. Xiao**. SUN RGB-D: A RGB-D scene understanding benchmark suite. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 567–576 (pages 12, 16, 89, 95, 99, 140).

[167] **S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser**. Semantic Scene Completion from a Single Depth Image. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pages 11, 141, 142, 193).

[168] **J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller**. Striving for Simplicity: The All Convolutional Net. In: *ICLR (workshop track)*. 2015 (page 46).

[169] **N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov**. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435 (pages 46, 89).

[170] **F. Steinbruecker, C. Kerl, J. Sturm, and D. Cremers**. Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. Sydney, Australia, 2013 (pages 5, 8).

[171] **F. Steinbruecker, J. Sturm, and D. Cremers**. Real-Time Visual Odometry from Dense RGB-D Images. In: *Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV)*. 2011 (page 8).

[172] **M. Stinchcombe and H. White**. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In: *International 1989 Joint Conference on Neural Networks*. 1989, 613–617 vol.1 (page 42).

[173] **J. Straub, G. Rosman, O. Freifeld, J. J. Leonard, and J. W. Fisher**. A mixture of manhattan frames: Beyond the manhattan world. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 3770–3777 (page 144).

[174] **J. Stückler and S. Behnke**. Integrating Depth and Color Cues for Dense Multi-Resolution Scene Mapping Using RGB-D Cameras. In: *proc. of IEEE International Conference on Multisensor Fusion and Information Integration*. MFI. Sept. 2012 (pages 83, 84).

[175] **J. Stückler, B. Waldvogel, H. Schulz, and S. Behnke**. Dense Real-Time Mapping of Object-Class Semantics from RGB-D Video. In: *J. of Real-Time Image Processing* (2015) (pages 5, 107).

[176] **J. Stückler and S. Behnke**. Orthogonal wall correction for visual motion estimation. In: *proc. of International Conference on Robotics and Automation (ICRA)*. 2008 (page 73).

[177] **J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers**. A Benchmark for the Evaluation of RGB-D SLAM Systems. In: *proc. of International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2012 (page 82).

[178] **H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller**. Multi-view convolutional neural networks for 3d shape recognition. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. 2015 (page 107).

[179] **R. W. Sumner, J. Schmid, and M. Pauly**. Embedded Deformation for Shape Manipulation. In: *ACM Transactions on Graphics* 26.3 (July 2007). ISSN: 0730-0301 (page 8).

[180] **I. Sutskever, J. Martens, G. Dahl, and G. Hinton**. On the Importance of Initialization and Momentum in Deep Learning. In: *proc. of International Conference on Machine Learning (ICML)*. Vol. 28. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, pp. III-1139–III-1147 (page 27).

[181] **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 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 00. June 2015, pp. 1–9 (pages 9, 49).

[182] **Y. Taguchi, Y.-D. Jian, S. Ramalingam, and C. Feng**. Point-plane SLAM for hand-held 3D sensors. In: *proc. of International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 5182–5189 (pages 9, 73, 83).

[183] **T. Tieleman and G. Hinton**. *RmsProp: Divide the gradient by a running average of its recent magnitude*. Tech. rep. COURSERA: Neural Networks for Machine Learning, 2012 (page 27).

[184] **A. J. B. Trevor, J. G. Rogers, and H. I. Christensen**. Planar surface SLAM with 3D and 2D sensors. In: *proc. of International Conference on Robotics and Automation (ICRA)*. May 2012, pp. 3041–3048 (pages 9, 73).

[185] **B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon**. Bundle adjustment - a modern synthesis. In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372 (page 9).

[186] **J. Valentin, V. Vineet, M.-M. Cheng, D. Kim, J. Shotton, P. Kohli, M. Niessner, A. Criminisi, S. Izadi, and P. Torr**. SemanticPaint: Interactive 3D Labeling and Learning at Your Fingertips. In: *ACM Transactions on Graphics* 34.5 (Nov. 2015), 154:1–154:17. ISSN: 0730-0301 (page 139).

[187] **S. Wager, S. Wang, and P. S. Liang**. Dropout Training as Adaptive Regularization. In: *proc. of Advances in Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 2013, pp. 351–359 (page 46).

[188] **W. Wang, H. Pottmann, and Y. Liu**. Fitting B-spline Curves to Point Clouds by Curvature-based Squared Distance Minimization. In: *ACM Transactions on Graphics* 25.2 (Apr. 2006), pp. 214–238. ISSN: 0730-0301 (page 147).

[189] **T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald**. Robust Real-Time Visual Odometry for Dense RGB-D Mapping. In: *proc. of International Conference on Robotics and Automation (ICRA)*. To appear. Karlsruhe, Germany, May 2013 (page 64).

[190] **T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald**. Kintinuous: Spatially Extended KinectFusion. In: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. July 2012 (pages 7, 15, 56, 167, 170).

[191] **T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger**. ElasticFusion: Real-Time Dense SLAM and Light Source Estimation. In: *International Journal of Robotics Research (IJRR)* (2016) (pages 7, 8, 107, 141).

[192] **T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. Mcdonald**. Real-time Large-scale Dense RGB-D SLAM with Volumetric Fusion. In: *International Journal of Robotics Research (IJRR)* 34.4-5 (Apr. 2015), pp. 598–626. ISSN: 0278-3649 (pages 7, 8, 38, 39, 83, 84, 178).

[193] **Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao**. 3D ShapeNets: A deep representation for volumetric shapes. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1912–1920 (pages 11, 193).

[194] **Z. Wu, C. Shen, and A. van den Hengel**. Wider or Deeper: Revisiting the ResNet Model for Visual Recognition. In: *CoRR abs/1611.10080* (2016) (page 107).

[195] **J. Yan, Y. Yu, X. Zhu, Z. Lei, and S. Z. Li**. Object detection by labeling superpixels. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 5107–5116 (page 123).

[196] **K. Yin, H. Huang, H. Zhang, M. Gong, D. Cohen-Or, and B. Chen**. Morfit: Interactive Surface Reconstruction from Incomplete Point Clouds with Curve-Driven Topology and Geometry Control. In: *ACM Transactions on Graphics (Special Issue of SIGGRAPH Asia)* 33.6 (2014), Article 202 (pages 141, 147).

[197] **F. Yu and V. Koltun**. Multi-Scale Context Aggregation by Dilated Convolutions. In: *proc. of International Conference on Learning Representations (ICLR)*. 2016 (pages 44, 107, 122, 123).

[198] **C. Yuksel, J. Keyser, and D. H. House**. Mesh colors. In: *ACM Transactions on Graphics* 29.2 (2010), 15:1–15:11 (pages 162–164).

[199] **M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus**. Deconvolutional networks. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2010, pp. 2528–2535 (pages 10, 44).

[200] **M. Zeiler and R. Fergus**. Stochastic pooling for regularization of deep convolutional neural networks. In: *proc. of International Conference on Learning Representations (ICLR)*. 2013 (page 109).

[201] **M. D. Zeiler**. ADADELTA: An Adaptive Learning Rate Method. In: *arXiv preprint* (2012). arXiv: 1212.5701 (page 27).

[202] **M. D. Zeiler and R. Fergus**. Visualizing and Understanding Convolutional Networks. In: *proc. of European Conference on Computer Vision (ECCV)*. Vol. 8689. Lecture Notes in Computer Science. Zurich, Switzerland: Springer, Sept. 2014, pp. 818–833 (pages 44, 89).

[203] **M. D. Zeiler, G. W. Taylor, and R. Fergus**. Adaptive Deconvolutional Networks for Mid and High Level Feature Learning. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2018–2025. ISBN: 978-1-4577-1101-5 (pages 10, 44, 46, 123, 129).

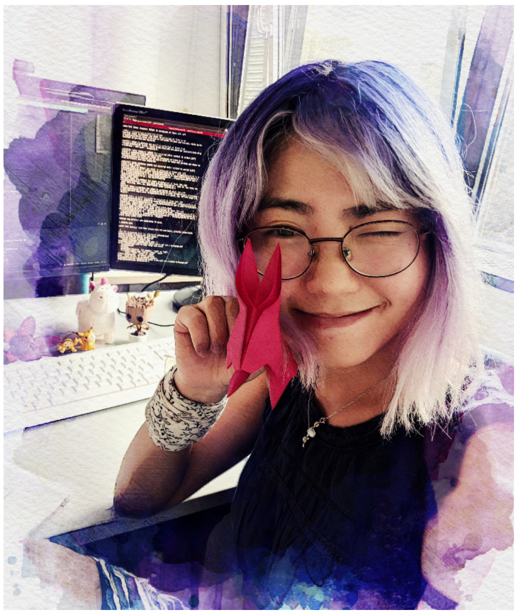
[204] **B. Zeisl, K. Kser, and M. Pollefeys**. Automatic Registration of RGB-D Scans via Salient Directions. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. Dec. 2013, pp. 2808–2815 (page 12).

[205] **H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia**. Pyramid Scene Parsing Network. In: *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 6230–6239 (pages 121, 124, 129, 130).

[206] **S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr**. Conditional Random Fields as Recurrent Neural Networks. In: *proc. of IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, Dec. 2015, pp. 1529–1537 (pages 88, 89, 122, 124).

[207] **Q.-Y. Zhou and V. Koltun**. Dense Scene Reconstruction with Points of Interest. In: *ACM Transactions on Graphics* 32.4 (July 2013), 112:1–112:8. ISSN: 0730-0301 (page 82).

Curriculum Vitea



Lingni Ma, was born in Sichuan, China. In 2010, she obtained her bachelor degree from Zhejiang University in China, from Electrical Engineering, majored in power electronics and very large-scale integrated circuits. From 2010 to 2012, Lingni studied in Technische Universiteit Eindhoven (TU/e) with full scholarship from the Talented Student Program in the Netherlands. She received the master degree of science in 2012 with cum laude from Electrical Engineering. After graduation, she worked in the video coding and architecture group at TU/e as scientific researcher under the supervision of Prof. Dr. Peter H. N. de With. From 2015, she started working as PhD researcher at Technische Universität München (TUM) in Germany. There, she joined the computer vision and artificial intelligence group at Computer Science Department, under the supervision of Prof. Dr. Daniel Cremers. During 2017 and 2018, Lingni did two research internship with the Surreal team at Facebook Reality Lab in Seattle, United States, under the supervision of Prof. Dr. Richard Newcombe. Lingni's research focus on visual SLAM, semantic understandings and deep learning. During her PhD study, she has worked on multiple aspects of semantic RGB-D SLAM, which has translated to 12 international peer-reviewed publications for conferences and journals. The algorithms she developed have been integrated into the Google Tango tablet and the Facebook 3D annotation tool. In 2018, Lingni will be joining Facebook Reality Lab as research scientist.