2019

Visual Perception For Robotic Spatial Understanding

Jason Lawrence Owens
University of Pennsylvania, jlowens@gmail.com

Visual Perception For Robotic Spatial Understanding

Abstract
Humans understand the world through vision without much effort. We perceive the structure, objects, and people in the environment and pay little direct attention to most of it, until it becomes useful. Intelligent systems, especially mobile robots, have no such biologically engineered vision mechanism to take for granted. In contrast, we must devise algorithmic methods of taking raw sensor data and converting it to something useful very quickly. Vision is such a necessary part of building a robot or any intelligent system that is meant to interact with the world that it is somewhat surprising we don't have off-the-shelf libraries for this capability.

Why is this? The simple answer is that the problem is extremely difficult. There has been progress, but the current state of the art is impressive and depressing at the same time. We now have neural networks that can recognize many objects in 2D images, in some cases performing better than a human. Some algorithms can also provide bounding boxes or pixel-level masks to localize the object. We have visual odometry and mapping algorithms that can build reasonably detailed maps over long distances with the right hardware and conditions. On the other hand, we have robots with many sensors and no efficient way to compute their relative extrinsic poses for integrating the data in a single frame. The same networks that produce good object segmentations and labels in a controlled benchmark still miss obvious objects in the real world and have no mechanism for learning on the fly while the robot is exploring. Finally, while we can detect pose for very specific objects, we don't yet have a mechanism that detects pose that generalizes well over categories or that can describe new objects efficiently.

We contribute algorithms in four of the areas mentioned above. First, we describe a practical and effective system for calibrating many sensors on a robot with up to 3 different modalities. Second, we present our approach to visual odometry and mapping that exploits the unique capabilities of RGB-D sensors to efficiently build detailed representations of an environment. Third, we describe a 3-D over-segmentation technique that utilizes the models and ego-motion output in the previous step to generate temporally consistent segmentations with camera motion. Finally, we develop a synthesized dataset of chair objects with part labels and investigate the influence of parts on RGB-D based object pose recognition using a novel network architecture we call PartNet.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Kostas Daniilidis

Keywords
calibration, computer vision, mapping, object parts, object recognition, visual odometry

Subject Categories
Artificial Intelligence and Robotics | Computer Sciences | Robotics

VISUAL PERCEPTION FOR ROBOTIC SPATIAL UNDERSTANDING

Jason Owens

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2019

Supervisor of Dissertation:

Kostas Daniilidis, Ruth Yalom Stone Professor
Department of Computer and Information Science

Graduate Group Chairperson:

Rajeev Alur, Zisman Family Professor
Department of Computer and Information Science

Dissertation Committee:
Camillo J. Taylor, Professor, Department of Computer and Information Science

Jianbo Shi, Professor, Department of Computer and Information Science

Jean Gallier, Professor, Department of Computer and Information Science

Nicholas Roy, Professor, Computer Science and Artificial Intelligence Lab, MIT

VISUAL PERCEPTION FOR ROBOTIC SPATIAL UNDERSTANDING

To Nettie, J, M, and E

Acknowledgments

The completion of my PhD would not have been possible without the support, understanding, and patience of my amazing wife Nettie and my children J, M and E. It is to them that I owe my deepest gratitude. So far, my children have endured most of their living memory with Daddy "going to school," and I am happy to finally begin the next chapter of our lives together!

I am very thankful for the support of my parents and my in-laws. My parents worked hard to give me the best opportunities and encourage me at every step. Throughout my entire trip at Penn, my in-laws have been there for me and my wife and kids, and have helped to provide space, time, food, and adult beverages when needed most. I am grateful and humbled to have such a caring family.

Many of my colleagues at ARL have been with me the entire time; encouraging, listening, commiserating, and otherwise tolerating my ups and downs through this degree. I especially thank Mr. Phil Osteen for being the most patient, thoughtful and supportive collaborator and friend. Not only do most of my papers have his name on them, but he often provided the steady head that helped us get through some of the most challenging deadlines, no matter what the cost. I also thank my other colleagues and friends that have been there to listen to my stories and provide support, including (but not necessarily limited to) Dr. MaryAnne Fields, Dr. Chad Kessens, Mr. Jason Pusey, Mr. Ralph Brewer, and Mr. Gary Haas.

I would like to thank Dr. Drew Wilkerson for strongly encouraging me to use the opportunities through the DOD and the Army Research Laboratory to pursue this education and degree. I must also thank Dr. Jon Bornstein for continuing to support me as my supervisor at ARL and the one who had to sign the most SF182s. Mr.

Geoff Slipher continued the tradition, and helped me get through the final hurdles; I am grateful for his consideration and understanding during this final year. Finally, I must thank ARL as an organization for the opportunity and privilege of attending a school such as the University of Pennsylvania; this has truly been a once in a lifetime experience.

I thank Dr. Richard Garcia for his friendship, support, and our drives to and from Philadelphia to attend Prof. Taskar's (*Requiescat in pace*) machine learning class.

I thank LTC Christopher Korpela, PhD. for crashing on my couch in Philly during *his* degree, and otherwise being a good friend and role model for getting $#!+ done.

What would time at the GRASP lab be but for my fellow students grinding through their degrees? I would like to thank Christine Allen-Blanchette for our many office discussions ranging over a wide variety of topics; always interesting, always stimulating, usually too short :-) I am grateful for my discussions with Matthieu Lecce; he helped me see I'm not the only one. I am very grateful for Kostas' Daniilidis Group discussions; the brainpower in that group is phenomenal. So many great explanations and just general geeking out.

I would like to thank the professors at Penn I've had the privilege to learn from and interact with, both in my capacity as a student and ARL researcher. I specifically thank Prof. C.J. Taylor for chairing my dissertation committee and always asking the hardest questions. I'm grateful to Prof. Jean Gallier for participating in my committee, and for the many enjoyable and amusing discussions across the hallway. I thank Prof. Jianbo Shi for offering his time for both technical and personal discussions, including several tips to help my kids do their piano practice. While not a Penn professor, I also must thank Prof. Nick Roy from MIT for being a part of my committee, and for helping me *recognize* the work I've done.

Last, but most definitely not least, I must thank my adviser and friend, Prof. Kostas Daniilidis, who patiently supported me, argued for me, and gently guided me through my stubborness for my many years in this program. He did what he could to pull me along, but I was often too hard-headed to comply; while I'm sure there is a part of him that will be glad to see me go (in a good way), I hope we can continue

to work together no matter what our respective circumstances might be. Thank you, Kostas!

For those whose names I left out (and I know there are many), I most sincerely thank you. Keep working hard and always do the right thing. But never forget to "take it easy."

ABSTRACT

VISUAL PERCEPTION FOR ROBOTIC SPATIAL UNDERSTANDING

Jason Owens

Kostas Daniilidis

Humans understand the world through vision without much effort. We perceive the structure, objects, and people in the environment and pay little direct attention to most of it, until it becomes useful. Intelligent systems, especially mobile robots, have no such biologically engineered vision mechanism to take for granted. In contrast, we must devise algorithmic methods of taking raw sensor data and converting it to something useful very quickly. Vision is such a necessary part of building a robot or any intelligent system that is meant to interact with the world that it is somewhat surprising we don't have off-the-shelf libraries for this capability.

Why is this? The simple answer is that the problem is extremely difficult. There has been progress, but the current state of the art is impressive and depressing at the same time. We now have neural networks that can recognize many objects in 2D images, in some cases performing better than a human. Some algorithms can also provide bounding boxes or pixel-level masks to localize the object. We have visual odometry and mapping algorithms that can build reasonably detailed maps over long distances with the right hardware and conditions. On the other hand, we have robots with many sensors and no efficient way to compute their relative extrinsic poses for integrating the data in a single frame. The same networks that produce good object segmentations and labels in a controlled benchmark still miss obvious objects in the real world and have no mechanism for learning on the fly while the robot is exploring. Finally, while we can detect pose for very specific objects, we don't yet have a mechanism that detects pose that generalizes well over categories or that can describe new objects efficiently.

We contribute algorithms in four of the areas mentioned above. First, we describe a practical and effective system for calibrating many sensors on a robot with up to 3 different modalities. Second, we present our approach to visual odometry and mapping that exploits the unique capabilities of RGB-D sensors to efficiently build detailed representations of an environment. Third, we describe a 3-D over-segmentation technique that utilizes the models and ego-motion output in the previous step to generate temporally consistent segmentations with camera motion. Finally, we develop a synthesized dataset of chair objects with part labels and investigate the influence of parts on RGB-D based object pose recognition using a novel network architecture we call PartNet.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The story I tell here extends all the way back to my childhood. Robots and artificial intelligence fascinated me. I had already been exposed to computers (an Apple IIc and an Apple Macintosh 512K), and wondered how they worked. When I learned of the concept of programming, the ability to make a computer do what you told it to do, I knew I wanted to do it. Some years later, I saw some robots in a magazine—they were Rodney Brooks' and his students' creations—and on a particularly nice day in SoCal I wondered how it would be possible to make an engineered machine behave intelligently. Of course, I really had no idea of the complexity and state-of-the-art at the time, but it looked like Brooks had done something amazing[1]. The question has remained fixed in my mind since that particular sunny day in front of my parents' house.

When I first started doing research related to robots, I was, therefore, particularly interested in how one might get them to have intelligent behavior. What that really meant was unclear, but it involved some combination of "moving around without running into things" and actually "accomplishing useful goals for humans and Soldiers." My initial conception developed into an interest in the topic of autonomous mental development: i.e., the capability of a robot to learn how to function for one or more tasks through interaction with the environment and without explicit programming

[1] I do believe Brooks (and his students) did accomplish something amazing, but it wasn't exactly what I was thinking about.

for any specific task. I wanted to (ultimately) develop a general "robot brain[2]."

As I began to investigate this line of research, I became frustrated with the state of the art in robot perception and, for lack of a better word, a robot's **understanding** of the world. My concept of a robotic brain (at the time) hinged on the idea that the world would be mostly observable and symbolic. This belief showed my utter naiveté at the time. The more I studied the literature, the more I came to understand that getting an embodied robot to function the way I envisioned would require some significant improvement to how it could use vision (as well as other modalities) to interpret the environment in a useable way. Again, I use another imprecise notion: "usability." I knew of the symbol grounding problem, and this was the exact embodiment of it. To really be able to become intelligent, the things and stuff in the environment must effectively become symbols that can be manipulated by an algorithm, and the symbols must not stand alone (to borrow Harnad's nomenclature [80], they should be related to both iconic and categorical representations that themselves can be manipulated). I assumed this algorithm would be complex, and the symbols would also be complex, but I had no clue how to get from pixels or point clouds to symbols. Perhaps needless to say, the goal of developing any version of a robot brain during my studies was postponed, as it became apparent that we (as researchers, and as a field) had more pressing problems on our hands.

Why is perception so important for robotics? First, I want to be precise about my use of the term "robotics." Robotics is a giant field, and covers many areas from what I would call unintelligent automation through human-like embodied intelligent agents. Throughout this document, I will use the terms 'robot', 'agent', and 'embodied intelligent agent' interchangeably, but in all cases I am referring to the latter unless explicitly stated. So, why is perception so important to *embodied intelligent agents*? They need the ability to intelligently see and understand the environment in order to participate effectively in the kinds of activities most useful to humans[3]. This desired ability is usually called perception, and is in contrast to some other vision tasks that

[2]Yes, I have very ambitious and often unrealistic goals. It is a problem I'm trying to work on.

[3]Especially the dirty, dangerous and/or defensive kind, but also the assistive, supportive and/or dull kind.

may not strive to enable the same level of intelligence. I also happen to believe that a robot with sufficient perception capabilities is a prerequisite to an *intelligent* robot, and that intelligence is required to have effective perception. They really go hand in hand. While it is **not** the case that perception is the only thing holding us back, it is an integral part of even the simplest system, and we must do everything we can to enable this capability.

What do I mean by perception? As I mentioned already, a robot capable of perception must "see and understand the environment." This is a very compact way to express a lot of ideas, so let's unpack it. Seeing is the ability to separate, pick out, and represent in some way or another, the important things in the environment. Understanding is harder to define, but we interpret it as the ability to transform what is seen into a *meaningful* and *useful* representation of the world. By "meaningful", we demand that the robot does more than determine a simple 1-1 mapping between sensed data (e.g., pixels) and labels; instead, it implies the association of **related** information such as experiences, similarities, differences and affordances as well as a deeper recognition of structure (hierarchical, spatial, temporal) and the ability to reason about what is sensed. All of this is necessary to make inferences about the nature and purpose of the environment and the objects in it on a level similar to a human. By "useful representation" we imply that there is more to the set of pixels or points or n-dimensional vectors that are output by an algorithm. Certainly, these elements may be part of the output, or even an integral component in the representation itself, but they must be part of a larger whole that can be operated on at multiple symbolic levels. Finally, the environment is everything around the robot, including everything that can change over time and space.

To go beyond data transformation and interpret information in relation to past experience and knowledge implies the need for reasoning capabilities typically considered outside the realm of computer vision (thus, my assertion that intelligence and perception are tightly related). However, I believe the following high-level capabilities are necessary for the task:

1. perceiving self motion, the motion of other entities in the environment, as well

as the structure of the environment;

2. separating things from stuff [85];

3. learning object instances and categories, as well as perceiving the context induced by the spatial relationships and configurations of the objects;

4. generalizing perceptual capability from previous experience.

These are not new ideas. However, they do provide the goals that motivate the development of algorithms we can use to compose behaviors to achieve this functionality.

Most of the approaches in this dissertation involve an abstraction that raises the level of semantic meaning of data, which corresponds well with the long-term goal of pursuing intelligent perception. For example, the multi-sensor graph calibration utilizes geometric representation and relationships between the sensed calibration object from multiple sensors in order to relate the data and perform the optimization. The primary goal of ego-motion estimation and the environment mapping was to use surfels that contain more information than just points in order to aid the alignment and mapping process. In temporal segmentation, we take point clouds and turn them into segments using similarities in their spatial structure so that you could work with larger logical chunks of the cloud instead of individual points or surfels within the cloud. Finally, we tackle parts of objects. Here, it apparently looks like we're taking a step down, and it is true in one sense; however, we are really interested in gaining *more* information about objects by understanding their composition and recognizing their pose.

1.1 Motion and the environment

To learn in the way I have described, I believe intelligent systems that are meant to interact with humans and the real world must actually be embedded in the real world. Successfully learning to understand the environment and the objects in it

will not happen by just looking at static pictures[4]. There must be understanding of how the world changes through differing viewpoints, how objects move and can be manipulated, and how the environment is structured by nature and humans (Grauman speaks directly of these topics in her latest talks on "Embodied Visual Perception" [75]). To address capability 1 above, sensors must be mounted on a mobile platform and must capture the true dynamic nature of the environment. This is in contrast to much of the work in simultaneous localization and mapping (SLAM) and environment mapping that tends to ignore the dynamic nature of the world. It is a hard problem, but all the more important on account of that. A dynamic world gives significant cues for object segmentation, 3 dimensional structure, and deformable objects.

We encounter our first obstacle after mounting multiple sensors on a robot. To process the data in a coherent manner, or more specifically, in the same coordinate frame of reference (usually with one sensor or the robot platform serving as the origin), we must know *where* the sensors are with respect to one another. This seems like it should be a trivial problem: we mounted them on the robot, therefore we should know where they are! Unfortunately, it is not that simple; first, every sensor is different and every mounting configuration is different; second, we do not always know the sensor's origin, even if we assume or know it is intrinsically calibrated[5]. To begin to collect data from a robot with multiple sensors, whether all of one type (e.g., all cameras) or of multiple modalities (e.g., a camera, a two-dimensional (2D) laser scanner, and a three-dimensional (3D) sensor), we must find the extrinsic calibration of the entire system: the relative pose of each sensor with respect to some origin. Chapter 4 discusses one approach to this challenge, and presents the only system we are aware of that solves the problem for a statically mounted set of sensors with multiple modalities and with very few requirements.

Having a robot with calibrated sensors means we can collect the sensor data into

[4]I'm looking at you, ImageNet. While this challenge has done much to spur development in algorithms for large scale recognition, it's applicability to intelligent embodied systems is limited by it's lack of focus on actually understanding the visual world; instead it focuses on generating mappings between pixels and labels without any accessible, underlying model.

[5]Third, I only consider rigidly mounted sensors in this dissertation. We are contemporaneously beginning to consider actuated sensors as an extension of the graph optimization framework presented in Chapter 4.

a common coordinate frame. Now we encounter a second obstacle: it would be difficult for our embodied system to move around and manipulate the world without understanding how it is structured and how it operates, at least at the local level (i.e., the portions of the environment it can currently see with the sensors). While I strongly believe much of this knowledge needs to be represented in a shared and consistent fashion, I also believe that larger problems must be broken down into chunks; therefore I separate the structure from the operation, and examine algorithms for ego-motion estimation and detailed environment mapping. Chapter 5 discusses our approach for using RGB-D cameras to capture accurate ego-motion and Chapter 6 uses the ego-motion estimates to build up maps of the environment. In these chapters, we will also discuss how these components may be used for extending the capability of robots in the future.

1.2 Things and stuff

Knowing the structure of the world is only one part of the challenge. Efficient map representations and planning algorithms may allow a robot to navigate the world, but how will it begin to learn about the objects in the world? There must be some way to partition the world into separate objects (usually the *things* we are interested in) and the other *stuff* that the world is made of (e.g., walls, floors, grass, sky). Achieving capability 2 above is the job of a segmentation and object proposal algorithm. While humans respond well to Gestalt cues (such as symmetry, continuity, common fate) [108] and some have made efforts to incorporate these cues into algorithms for object segmentation [101], [111]–[114], we focus on the under-explored topic of over-segmenting a scene in a temporally consistent way. An over-segmentation is often used as a pre-processing step, and helps later algorithms by clustering primitive perceptual data into larger chunks that are typically spatially, visually, and geometrically coherent. While many over-segmentation algorithms operate in image space only, in Chapter 7, we modify an existing algorithm that operates on 3D point clouds to support extending segmentations to neighboring frames based on an accurate ego-motion

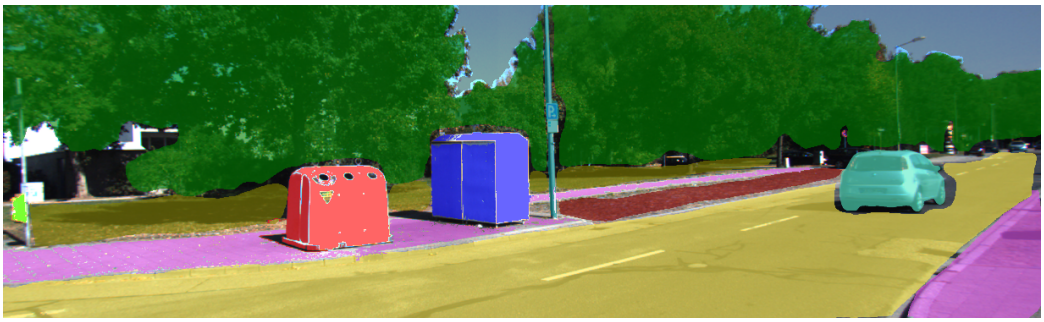


Figure 1.1 A near-ideal segmentation of a street scene.

estimation and frame integration algorithm. This builds on the work from the previous section, and would not be possible without a good estimation of the sensor motion.

Segmenting a scene (or the environment), the particular partitioning of what is seen into chunks (hopefully as close to an ideal segmentation as possible, see Fig. 1.1), is a prerequisite to being able to discover and learn about objects. While there are attempts to do this in concert with object recognition [26], [82], [174], we believe the best approach is to separate the task of segmentation and object *proposals* from recognition, and ultimately focus on an iterative interplay between the algorithms. Without this, it is not clear how to dynamically extend a recognition system with new information on the fly.

1.3 Objects

It is clear that a robot cannot do much useful work without being able to first detect, identify, and describe objects, and second, manipulate those objects. Objects are a fundamental aspect of the world. They are the *things* we want to manipulate or observe as well as the *stuff* that surrounds us. An intelligent embodied system must be able to see objects in such a way as to determine their location, identity (up to some extent), and category as well as whether and how they can be manipulated (i.e., their affordances). If a robot did not possess such a capability, most things would simply be implictly classified as an obstacle, or otherwise occupied space.

Objects may be recognized in two distinctly different ways: as particular instances

of a specific object (e.g., a box of one kind of cereal), or as an instance of category, whether specific or general. The first kind of recognition is used extensively in robots, to handle a small sampling of real-world objects in a very specific way, by both representing appearance and geometry of the object in order to detect presence and pose simultaneously. However, this means that if it recognizes one cereal box, then it may not recognize another, even if they are very similar in many ways: size, geometry, general appearance (e.g., a lot of bright colors). The second kind of recognition, on the other hand, is, by definition, more general, and refers to the ability to recognize that a box of any brand of cereal is, in fact, a box of cereal. In many cases, however, categorical object recognition (the most common type) is unable to recognize specific instances, or even determine that box A and box B are the same kind of cereal. Interestingly, humans perform both of these tasks pretty well, and it may prove to be hard for robots to interact seamlessly and efficiently with their biological collaborators without a similar level of proficiency.

The question remains: how do we induce an algorithm to "recognize" anything? Research to address this question has been going on for over 50 years and we have developed specific algorithms for detection ("am I looking at an object of instance i or category k?"), localization ("where is the object and what is its pose"), and recognition ("find and identify all known objects") [3]. Detection, localization and recognition all rely on *some form* of matching input data to previously learned models. As part of that input data, we have strong evidence that the human visual system does extract low level features early in the visual process [137], [207]. One of the basic ideas behind the use of features is to find regions of interest in order to use them as descriptions in a model of the object or environment such that it is possible to match the input data to previously learned models, hopefully without a linear search.

While deliberately leaving the definitions of "feature", "match", "description", and "concept" vague, imagine we want to create a database of known objects for use in a robotic vision task. We would like to extract salient features that could be used to recognize these objects in new environments, where they may be seen in a variety of poses and distances from the sensor. In what ways can we detect and extract these

features to make the object detection and recognition task more effective? If we want to detect local features, we must consider a neighborhood surrounding a point to determine whether that point actually represents the location of a feature. How do we select the size of the neighborhood?

To answer some of these questions and work to address our previously stated capability 3, we propose to investigate convolutional networks that perceive additional information about objects in the form of parts and pose. Most objects are composed of more than one part, and often the parts may be individually recognizable components. If not individually recognizable, then the collection of parts may still provide additional useful information about the structure and pose of the object. While this approach may not directly address the instance recognition task mentioned above, if successful it could represent a significant improvement in pose estimation capability for general object categories. We describe our initial approach and results in Chapter 8.

1.4 Generalization

Generalization is, in some ways, the holy grail of artificial intelligence, and therefore most of its sub-fields, like robotic perception. We interpret the concept in this very simplified sense: the agent experiences something, extracts some useful information or knowledge . . . then experiences something else partly related and *applies* the previous information or knowledge to the new situation. In computer vision, a version of this is directly seen in the ability of a recognition system to train (experience) on many different instances of a particular category, say, chairs, and then recognize chair instances it has never seen before. In these cases, we would really like the system to be able to explain why this new object might be a chair, but we will hold off on that for now.

Another example of generalization capability is demonstrated in the areas of few-shot, one-shot, and even zero-shot learning. The idea here is that if you can teach a system how to learn within a domain, then showing it very few examples, or even

one example, should allow it to generalize to new instances. In the zero-shot case, **no** examples are shown, but instead some kind of semantic description ("large, white animal that lives in snowy regions") should be sufficient to recognize visual images of the object.

While this thesis does not directly address this specific capability directly, we believe the efforts towards part-based recognition systems and the increasing interest in symbol grounding and online learning from experience will make generalization an important aspect for every robotic system.

1.5 Publications

The work in this dissertation is based in part on several publications, listed below. Contributions from collaborating authors will be annotated in the relevant sections.

[156] P. R. Osteen, J. L. Owens, and C. C. Kessens, "Online egomotion estimation of RGB-D sensors using spherical harmonics," in *Robotics and Automation (ICRA), 2012 IEEE International Conference On*, IEEE, 2012, pp. 1679–1684.

[157] J. L. Owens, P. R. Osteen, and K. Daniilidis, "Temporally consistent segmentation of point clouds," in *SPIE DEFENSE+ Security*, International Society for Optics and Photonics, 2014, 90840H–90840H.

[158] ——, "MSG-Cal: Multi-sensor Graph-based Calibration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[159] J. Owens and M. Fields, "Incremental region segmentation for hybrid map generation," in *Army Science Conference*, 2010, pp. 281–286.

[161] J. Owens and P. Osteen, "Ego-motion and tracking for continuous object learning: A brief survey," US Army Research Laboratory, Aberdeen, MD, Technical Report ARL-TR-8167, Sep. 2017, p. 50.

[201] R. Tron, P. Osteen, J. Owens, and K. Daniilidis, "Pose Optimization for the Registration of Multiple Heterogeneous Views," in *Multi-View Geometry in Robotics at Robotics, Science, and Systems*, 2014.

Chapter 2

Related Work

In this section, I provide an overview of the related work for each algorithm I present in this thesis. While it is not meant to be an exhaustive exposition, I provide summaries and short discussions of the most relevant literature that have informed my work.

2.1 Multi-sensor calibration

The fundamental challenge in sensor calibration is determining associations between each sensor's data[1]. Recent calibration approaches use either known scene geometry (including specialized calibration targets), or attempt to calibrate in arbitrary environments. Systems that calibrate with no special scene geometry or calibration object, such as [125] and [134], require high quality inertial navigation systems (INS) to compute trajectory. Scaramuzza et al. are able to calibrate a 3D laser and a camera without a calibration object or inertial sensors, by having a human explicitly associate data points from each sensor [182]. In contrast, our calibration framework (Chapter 4) uses only the data coming from the sensors we wish to calibrate, and does not require human assisted data association.

Similar to the work of [127], [143], [154], [204], [233] and others, we also use a calibration object to facilitate automatic data association. To our knowledge, there has been little work in the calibration of an arbitrary number of various types of

[1]Content in this section has been adapted from the author's contribution in [158], ©2015 IEEE.

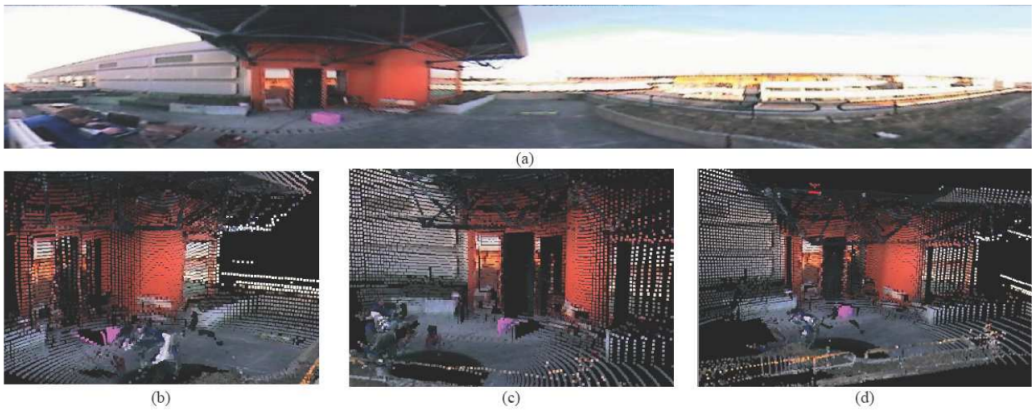


Figure 2.1 Scaramuzza et al. finds the extrinsic calibration between a 3D laser range finder and a camera, but requires a human to label correspondences. Image reproduced from [182], ©2007 IEEE.

sensors. Most work on calibration has been limited to either a single pair of different sensor types ([125], [127], [143], [154], [182], [204], [205], [233]), or multiple instances of a single type ([134], [142]). An exception is the work of Le and Ng [121], who also present a framework for calibrating a system of sensors using graph optimization, though their graph structure differs fundamentally from ours; they require that each sensor in the graph be a 3D sensor. Therefore, neither individual cameras nor 2D laser scanners are candidates in their calibration. Instead they must be coupled with another sensor (e.g., coupling two cameras into a stereo pair) that will allow the new virtual sensor to directly measure 3D information. Their results agree with ours, that using a graph formulation reduces global error when compared with incrementally calibrating pairs of sensors. Our approach is related to the graph optimization proposed in [208] where the focus is on solving non-linear least squares systems on a manifold. Our system generates initial estimates from pairwise solutions, and we report the results of calibrating a graph of five sensors with three different modalities.

One of the fundamental challenges in sensor calibration is determining associations between the sensors' data. Calibration approaches generally use one of two methodologies: using known scene geometry (including specialized calibration targets), or using arbitrary environments. Systems that calibrate multiple sensors using no special scene geometry or calibration objects, such as [125] and [134], require a high quality inertial navigation system (INS) to compute trajectory.

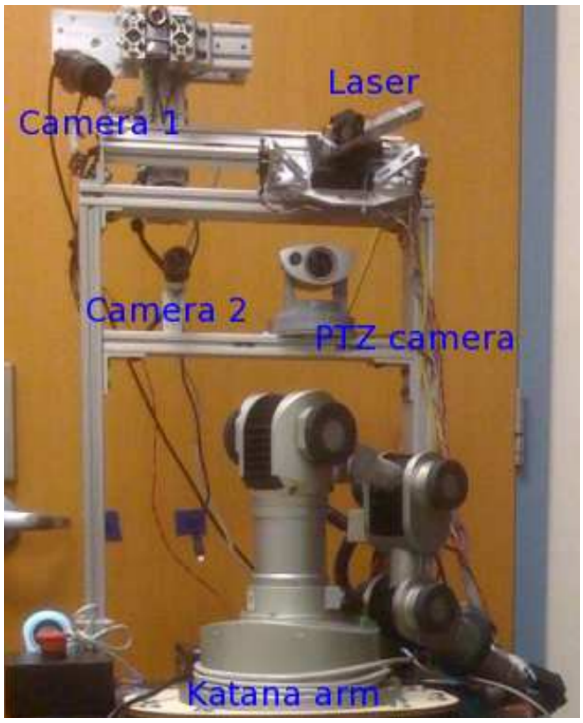


Figure 2.2 Le and Ng's work on calibration seems to be closest to ours (see Chapter 4). They calibrate multiple sensors in a graph; however, they require pairs of sensors to generate a single 3D observation. Image reproduced from [121], ©2009 IEEE.

Our work falls into the category of using a specialized calibration object to generate *managed* observations from sensors. This is similar to the work of [127], [143], [154], [204], [233] and others. To our knowledge, there has been little work in the calibration of an arbitrary number of multiple types of sensors.

2.2 Ego-motion and Mapping

Fundamentally, ego-motion from visual odometry is concerned with computing the motion of a sensor using visual data streaming from that sensor[2]. But **what** data from the sensor are we using? Following the classification of Engel et al. [46], the computational method for solving for incremental pose transformation between sensor frames can be described using 2 dimensions: the nature of the input and the density of the input. The first dimension describes what *values* are used in the computation; in visual odometry, the data provided by the sensor(s) are spatially organized

[2]Content from this section has been adapted from the author's contribution to [161].

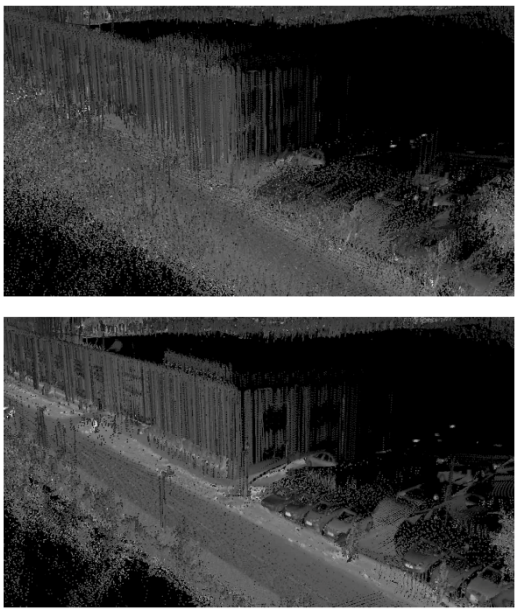


Figure 2.3 The calibration work of Maddern et al. calibrates 2D and 3D laser range finders to the robot frame using a point cloud quality metric, uses no calibration object, but relies on locally accurate ego-motion estimation. Image reproduced from [134], ©2012 IEEE.

photometric measurements of the amount of light registered by the sensor (i.e., the pixel values). If an algorithm uses this photometric information, then it is called a *direct* method. Alternatively, if there are derived values computed from the image, (e.g., keypoints, descriptors, and/or vector flow fields), then we call the method *indirect*. Note that since direct methods use the photometric information directly, the optimization models photometric noise, while the derived values in the indirect case are geometric in nature (points and vectors), and therefore the optimization models geometric noise.

The second dimension indicates how much information is used for the computation. If an algorithm attempts to use all the pixels (or as many as possible) from the input, then it is called a *dense* method. In contrast, a *sparse* method specifically uses a small subset of the pixels/points available, usually around 2 orders of magnitude less than than the number of pixels. Dense methods do not extract a discrete set of features, but instead rely directly on the data itself to compute an estimate of motion. The main idea is to use more of the image data than sparse methods with the goal of improving the camera and environment model motion estimates. In contrast, sparse methods

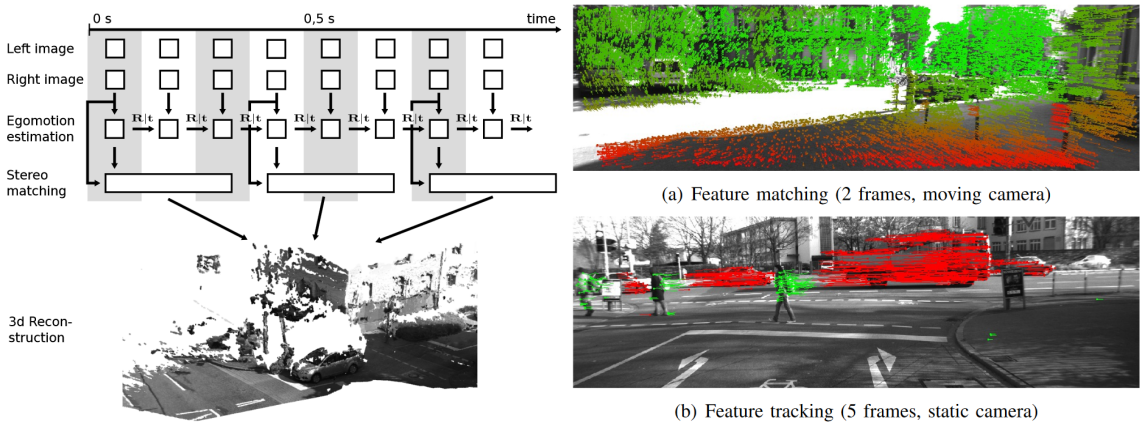
often use discrete feature keypoints (e.g., Scale Invariant Feature Transform (SIFT) [132], Oriented Rotated BRIEF (ORB) [177]). These keypoints, along with the data, serve as input to an algorithm that will produce a set of descriptors. The descriptors are representations of the input, designed to enable feature matching between frames by comparing the distance between pairs of descriptors.

For simplicity, we primarily focus on stereo, red, green, blue, and depth (RGB-D), and laser-based modalities that can directly compute 3D features within the environment, as they are the most useful for enabling accurate, scale drift-free maps and motion estimates. Monocular algorithms, while powerful and efficient, cannot compute the scale of the environment (although they can be filtered with other odometry methods that can). 2D laser scan-matching algorithms, while very popular on experimental robotic systems, are also insufficient for our needs, since they cannot compute full 6-DOF pose estimates and are easily confused by nonflat environments.

`libviso2` by Geiger et al. [70] is a simple but effective stereo visual odometry algorithm created by the group responsible for the KITTI benchmark suite [69]. It is a prime example of sparse indirect visual odometry methods (see Scaramuzza and Fraundorfer [181] for a nice overview of the approach). Ego-motion is computed using simple blob and corner-like features distributed over the full image, which are stereo matched between the left and right frame to compute 3D pose, and then temporally matched between successive frames to estimate motion (Fig. 2.4). Complex feature descriptors are not needed since there is an assumption that frames are temporally and spatially close (i.e., from a 30-Hz camera). Instead, a simple sum of absolute differences of an 11×11 pixel block around the keypoint is used as a descriptor distance method. Ego-motion estimation is implemented as a Gauss-Newton minimization of reprojection error:

$$\sum_{i=0}^{N} \left\| \mathbf{x_i}^{(l)} - \pi^{(l)}(\mathbf{X_i}, \mathbf{r}, \mathbf{t}) \right\|^2 + \left\| \mathbf{x_i}^{(r)} - \pi^{(r)}(\mathbf{X_i}, \mathbf{r}, \mathbf{t}) \right\|^2, \tag{2.1}$$

where $\mathbf{X_i}$ are the 3D points corresponding to the features $\mathbf{x_i^{(l)}}, \mathbf{x_i^{(r)}}$ in the left and right images, $\pi^{(l)}, \pi^{(r)}$ are the corresponding left and right projection functions, based



(a) Feature matching (2 frames, moving camera)

(b) Feature tracking (5 frames, static camera)

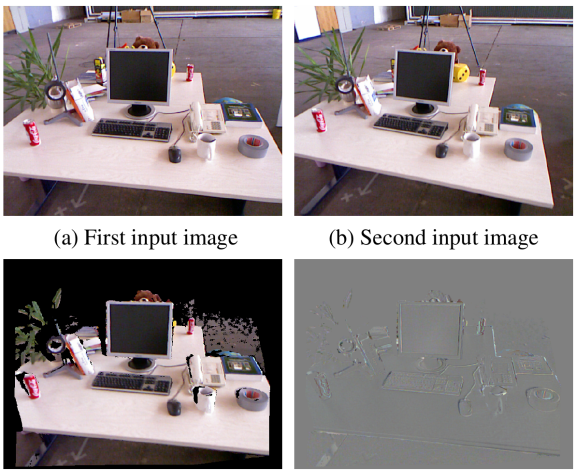
Figure 2.4 `libviso2` **visual odometry system, an example of a sparse, indirect visual odometry (VO) method. The left image illustrates the overall operation of the system (temporal ego-motion estimation, stereo matching and 3D reconstruction). The right image shows the matched features and their motion in 2 situations: (a) moving camera and (b) static camera. Reproduced from Geiger et al. [70], ©2011 IEEE.**

on intrinsic and extrinsic calibration of the cameras, and $\mathbf{r}, \mathbf{t}$ are the rotation and translation parameters being estimated.

Steinbrucker et al. [192] present DVO, a VO algorithm that is based on RGB-D sensors instead of stereo cameras. DVO eschews feature extraction in favor of a dense model in order to take advantage of as much of the image as possible. The algorithm utilizes a photometric consistency assumption to maximize the photoconsistency between 2 frames. Photometric consistency means that 2 images of the same scene from slightly different views should have the same measurements for pixels corresponding to a given 3D point in the scene. Steinbrucker et al. minimize the least-squares photometric error:

$$E(\xi) = \int_{\Omega} \left[I(w_{\xi}(x, t_1), t_1) - I(w_{\xi}(x, t_0), t_0) \right]^2 dx, \tag{2.2}$$

where ξ is the twist representing the estimated transformation, and the $w(x, t)$ function warps the image point x to a new image point at time t using the transformation ξ and the associated depth map. $I(x', t)$ is a function that returns the image intensity value at position x' at time t. Note how this represents a dense direct approach, integrating over the entire image domain (Fig. 2.5), while the sparse indirect approach in `libviso2` is a sum over only the extracted features. The 3D points in an RGB-D



(a) First input image (b) Second input image

(c) Warped second image (d) Difference image

Figure 2.5 An example of RGB-D image warping in DVO. (a-b) show the input images separated by time, while (c) shows the second image warped to the first, using the photometric error and the depth map to compute how the pixels should be projected into the warped frame. (d) illustrates the small error between (a) and (c). Reproduced from Steinbrucker et al. [192], ©2011 IEEE.

approach come directly from the produced depth map and do not require any explicit stereo computations. In practice, dense approaches vary in their actual density: RGB-D based dense approaches can be much more dense than monocular or stereo approaches, although in both cases they use more image data than a sparse approach. Unfortunately, the cost of higher density is more computation, so the tradeoff often depends on the accuracy required in the motion estimation.

The use of current RGB-D sensors has benefits and drawbacks: as active stereo devices, they do not require strong features in the environment to compute depth values, yielding dense depth images; however, they usually have relatively short range (relying on a pattern or measured time of flight from an infrared (IR) projector) and do not work well (if at all) in outdoor conditions[3].

Lidar odometry and mapping (LOAM [231]) and visual odometry-aided LOAM (V-LOAM [232]) are related ego-motion algorithms by Zhang and Singh that use 3D laser scanners (e.g., the Velodyne HDL-32 [206]) to compute both maps and motion estimates[4]. LOAM works (and works very well, see the KITTI [69] odometry bench-

[3]Newer RGB-D sensors seem to address this problem, with slightly longer ranges and claims of outdoor operation. We do not yet have direct experience to comment on these claims.

[4]LOAM and V-LOAM are not full SLAM algorithms, since they do not include loop closure and

marks) by carefully registering 3D point cloud features to estimate sensor motion at a high frame rate, and then integrating full undistorted point clouds into the map at lower frame rates while adjusting the sensor pose estimates. This algorithm can work with sweeping single-line laser scanners as well as full 3D scanners by estimating the pose transform between each line scan. If such a sensor is available, this algorithm produces some of the lowest error motion and pose estimates over large-scale paths. The primary drawback is that the code is no longer available in the public domain, since it has been commercialized as a stand-alone metrology product.[5]

Monocular visual odometry algorithms have recently become more popular on small robotic platforms due their extremely simple sensor arrangement (i.e., a single camera) and their ability to operate on mobile computing hardware [43], [184]. Several recent monocular algorithms (semi-dense visual odometry [49], Semi-direct Visual Odometry [SVO] [64], and Direct Sparse Odometry [DSO] [46]) demonstrate impressive mapping performance with low drift, all within a direct VO paradigm (i.e., they *directly* make use of the values from the camera and therefore optimize a photometric error). These algorithms operate by estimating semi-dense depth maps using strong gradients in the image along with temporal stereo (comparing 2 frames that differ in time, with the assumption that the camera was moving during the time period), and make extensive use of probabilistic inverse depth estimates. DSO is particularly interesting since it is a direct method using sparse samples (but no feature detection) and it exploits photometric sensor calibration to improve the robustness of the motion estimation over existing methods.

However, for monocular algorithms to be useful for typical robotics applications, there must be some way to estimate the scale of the observed features for map generation and motion control. A typical (but not singular) method for accomplishing this is by filtering the visual features with some other metric sensor such as an iner-

cannot perform global localization.

[5]Although the code was originally available as open source on Github and documented on the Robot Operating System (ROS) wiki [230], the repositories have since been removed. Fortunately, some forks of the repositories still exist, but the full ROS integration does not seem to exist. In addition, the code is undocumented, has hard coded transformation assumptions, and is not written for modularity or future programmers. However, it may provide sufficient guidance to those who wish to adapt and extend the algorithm in the future.

tial measurement unit (IMU). This can be considered a sub-field of visual odometry, typically called visual-inertial navigation. Since we assume we will have access to stereo, RGB-D, or 3D laser information, reviewing this topic is out of scope for this document. However, for more information, see Weiss' tutorial on "Dealing with Scale" presented at Conference on Computer Vision and Pattern Recognition (CVPR) 2014 [212].

2.2.1 Mapping

Many of the VO algorithms we discuss create local models of the environment to achieve more accurate motion estimates (i.e., odometry with less drift). It might be natural to assume these algorithms could be included as the front-end in a SLAM framework, and that assumption would be correct (see Cadena et al. [21] for a great overview of SLAM with a look toward the future). A SLAM front-end provides motion estimates (like what we get from VO), while a SLAM back-end optimizes a map given local constraints between sensor poses, observed landmarks, and global pose relationships detected as "loop closures". A loop closure is the detection of 2 or more overlapping observations, often separated considerably in time but not in space; see Fig. 2.6 for a simple illustration. For example, if a robot maps a room, exits to map another room or hallway, and then re-enters the first room, it should detect a loop closure by virtue of being in the same place it was before and therefore seeing the same landmarks. The term "loop" comes from the fact that the pose graph (where sensor poses are vertices, and edges represent temporal or co-observation constraints) forms a cycle or loop of edges and vertices when a revisitation occurs.

Stereo LSD-SLAM [48] is a camera-based algorithm that represents an extension of the LSD-SLAM [47] monocular algorithm to a stereo setting. LSD stands for large-scale direct SLAM and, as the name indicates, represents a direct, dense image-based VO algorithm integrated into a SLAM system handling mapping and loop closure. They build on the original algorithm by adding support for static stereo to estimate depth, requiring no initialization method (typically required for monocular VO algorithms) and obviating the need for the original scale-aware similarity transform

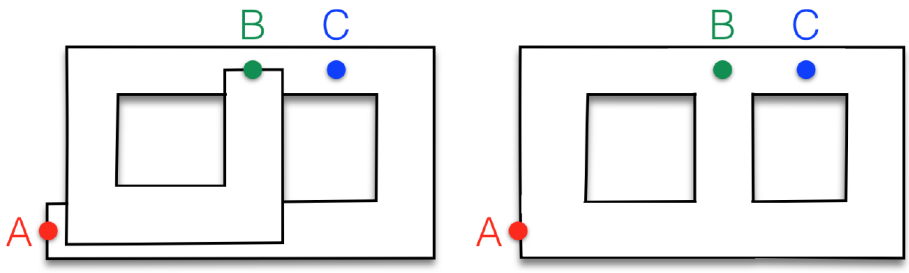


Figure 2.6 An example of the loop-closing concept in a simplified hallway. The left image shows a camera that has computed visual odometry and revisited both point A and point B without closing the loop. The right image, on the other hand, is the map result from a SLAM algorithm after detecting the loop closure and optimizing the map. Reproduced from Cadena et al.[21], ©2016 IEEE.

estimation. In addition, they add in a simple exposure estimation procedure to help counteract the effects of lighting changes in real-world environments. Their results are impressive but a little slow for larger image sizes (640×480 runs at only 15 Hz) if one is concerned about running in real-time on a robot. Currently, code is available for *only* the monocular version.

As an example of a hybrid-indirect VO methodology used in a SLAM framework, Henry et al. [87] was one of the first high-quality dense mapping algorithms to make use of RGB-D sensors like the Microsoft Kinect. The core algorithm is composed of a frame-frame iterative closest point (ICP) algorithm on the point cloud derived from the depth image, which is jointly optimized with sparse feature correspondences derived from the red, green, and blue (RGB) images. It uses both sparse and dense data, thus forming what we call a "hybrid" methodology. The ICP and sparse feature alignment are complementary: the former performs better with significant geometry complexity in the environment, even when featureless, while the latter supports alignment when there are features, but little to no geometry (e.g., walls and floors with posters or other textured objects). To generate higher-resolution clouds, they post-process the optimized map by integrating depth frames using a surfel-based approach [167] to better adapt regions with differing point density.

Another sparse-indirect VO method applied in a SLAM framework is ORB-SLAM2 [149], an extension of the original ORB-SLAM framework [148] to stereo and RGB-D cameras for metric scale. ORB-SLAM2 is a combination of existing techniques that

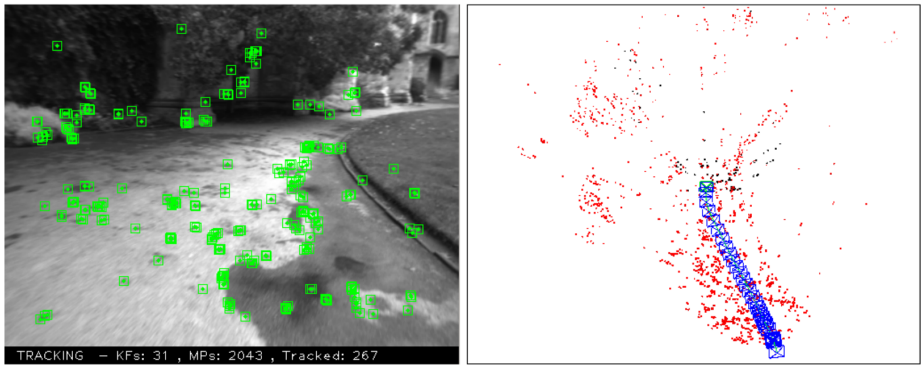


Figure 2.7 ORB-SLAM feature and map density. Note the sparsity of the feature-based map. Image reproduced from Mur-Artal et al. [148], ©2015 IEEE.

provide a robust SLAM system with the rare capability of reusing maps for localization in previously visited areas. As the name implies, Mur-Artal and Tardós make use of the ORB [177] feature detector/descriptor to detect and describe features in each frame (Fig. 2.7). Several aspects of this algorithm make it stand out: (1) both near and far features are included in the map, (2) the map is reusable for localization without mapping, and (3) there is no dependence on a graphics processing unit (GPU). While aspect (3) has obvious benefits (i.e., it can be adapted to run on systems without a GPU), the first 2 require some additional explanation. In most recent monocular VO approaches, feature depth (within a given keyframe) is parameterized as inverse depth. This makes it easy to incorporate points at larger distances, since $\frac{1}{d}$ for infinite d is just 0. Points at infinity or with uncertain depth are still useful for computing accurate rotations, even if they cannot be used to estimate translation. By incorporating near (i.e., depth computable from stereo or RGB-D) and far points, ORB-SLAM2 can generate better pose estimates using more information. Finally, while SLAM algorithms are used to build maps, many are unable to localize within those maps without generating new map data. This feature is particularly useful for a robotic system that may encounter new areas over time, but still navigate previously mapped regions repeatedly.

In the popular subfield of 3D reconstruction using RGB-D sensors, 2 recent methods exemplify the volumetric integration approach. Ever since the influential KinectFusion papers [100], [152] demonstrated the use of on-GPU truncated signed distance

function (TSDF) volumes, many researchers have expanded on and refined the approach [86], [122], [128], [193], [213], [215], [217]. We highlight 2 recent examples, both capable of extended model reconstruction (i.e., a larger volume than can fit in GPU memory) and both utilizing full RGB-D frame information.

ElasticFusion [216] computes a dense surfel model of environments, notably without using pose graph optimization. Instead, the system relies on frequent registration with both active (recently acquired and currently used for pose estimation) and inactive (local but older observations) portions of the map, the latter reflecting loop closures. Upon registration with inactive portions of the map, the algorithm induces a nonrigid deformation that helps to bring that portion of the map into alignment with the currently active version. It is this deformation process that obviates the need for a global graph optimization phase, with the assumption that the registration reduces the error in the model enough to not require the optimization. Indeed, the results reported show some of the lowest reconstruction errors on popular benchmark datasets. However, it should be noted that they do not show spaces larger than a large office environment, while previous approaches such as Kintinuous [213] show larger outdoor scenes and multiple indoor floors.

BundleFusion [38] aims to provide an all-around solution to real-time 3D reconstruction, with the unique capability of real-time frame deintegration and reintegration in the global volumetric model. A sparse to dense local pose alignment algorithm is used to improve the pose estimation (utilizing SIFT features for sparse correspondence) between the frames, while a global optimization is performed after scanning has ended. Like the previous approach, this algorithm produces very good dense scene models, but suffers from 2 main limitations: scene size (or recording time) is limited to a maximum of 20,000 frames (due the nature of their on-GPU data structures), and they require 2 powerful GPUs in parallel to achieve real-time performance, limiting application to larger robotic systems capable of carrying and powering twin GPUs.

2.2.2 Attributes

In this subsection, we identify salient properties of the algorithms reviewed in the egomotion survey. Table 2.1 compares selected algorithms according to the following attributes:

Input What is the input modality? ([Mono]cular camera, [Stereo] camera, [RGB-D], [2D] light detection and ranging (LIDAR), [3D] LIDAR)

Direct Direct versus indirect parameter estimation (i.e., photometric versus geometric optimization)

Dense Is the representation dense (versus sparse)?

Mapping Does the system generate maps?

Loop Closing Does the method handle place recognition for closing loops (revisited locations) in maps?

Known Scale Does the method produce maps with known scale (related to sensor modality)?

Persistent Can the existing implementation store new maps, and load and modify previously generated maps?

GPU Does the algorithm utilize/require a GPU?

Scalability Color saturation represents a comparative and qualitative estimation of the scalability of the algorithm.

Code Do the authors make source code available?

2.2.3 Challenges

These sections have covered the state of the art in ego-motion estimation, and mapping algorithms for a variety of tasks. Some of them perform very well in the environments (or datasets) they are tested in, but our primary concern is how they will

Table 2.1 An overview of selected ego-motion algorithms reviewed in this section and their attributes.

	Input	Direct	Dense	Mapping	Loop Closing	Known Scale	Persistent	GPU	Scalability	Code
viso2 [70]	Stereo	✗	✗	✗	✗	✓	✗	✗		✓
DVO [192]	RGB-D	✓	✓	✗	✗	✓	✗	✗		✓
LOAM [231]	2D/3D	✗	✗	✓	✗	✓	✗	✗		✓[a]
vLOAM [232]	2D/3D + Mono	✗	✗	✓	✗	✓	✗	✗		✗
SDVO [49]	Mono	✓	✓[b]	✗	✗	✗	✗	✗		✗
SVO [64]	Mono	✓[c]	✗	✗	✗	✗	✗	✗		✓
DSO [46]	Mono	✓	✗	✗	✗	✗	✗	✗		✓
SLSD-SLAM [48]	Stereo	✓	✓	✓	✓	✓	✗	✗		✗
LSD-SLAM [47]	Mono	✓	✓	✓	✓	✗	✗	✗		✓
RGBD-Mapping [87]	RGB-D	✗	✗	✓	✓	✓	✗	✗		✗
ORB-SLAM2 [149]	RGB-D or Stereo or Mono	✗	✗	✓	✓	✓	✓	✗		✓
ORB-SLAM [148]	Mono	✗	✗	✓	✓	✓	✗	✗		✓
KinectFusion [100], [152]	RGB-D	✓	✓	✓	✗	✓	✗	✓		✗[d]
VolRTM [193]	RGB-D	✓	✓	✓	✗	✓	✗	✗		✓
PatchVol [86]	RGB-D	✗	✓	✓	✓	✓	✗	✓		✗
Kintinuous [213], [217]	RGB-D	✓	✓	✓	✗	✓	✗	✓		✓
HDRFusion [128]	RGB-D	✓	✓	✓	✗	✓	✗	✓		✓
ElasticFusion [216]	RGB-D	✓	✓	✓	✗	✓	✗	✓		✓
BundleFusion [38]	RGB-D	✓	✓	✓	✗	✓	✗	✓		✓

[a] There are forked repositories, but the original has been removed.

[b] Only image regions with strong gradient in the motion direction are used.

[c] Semi-direct: direct methods are used for motion estimation, but features are extracted for keyframes and used for bundle adjustment.

[d] KinectFusion does not seem to have the original MS code published, but there is at least one open source implementation available through the Point Cloud Library (PCL).

perform in the context of an intelligent embodied agent. Our view of these agents places them along with other "life-long" learning systems; these are systems that should learn incrementally and adapt online over long periods of time. How will ego-motion estimation and mapping scale to handle experiences that range over larger and larger distances and more complicated, unstructured terrain? This section highlights some of the challenges we see when considering the integration of ego-motion estimation for a continuous object learner.

Many existing ego-motion algorithms can run continuously, specifically those based on visual odometry, providing information about motion within the local environment. However, the longer they run, the larger the pose error relative to ground truth will be without measuring against some external source. We take the position that objects

are *not* independent of their context (i.e., where they are, how they are used, and their relationship with other nearby objects). Therefore, it is important to place objects within an environmental context so they can be reobserved with confidence. Accurate local metric representations can provide this confidence, and visual odometry methods without this kind of mapping capability may not suffice.

SLAM, on the other hand, may provide the facilities needed to produce accurate local maps (see examples of this in section 6.2). However, many of these algorithms operate indoors (due to sensor or memory limitations) or on city streets; while they may be cluttered and complex, there is still significant structure in the environment. Army systems meant to operate in a variety of environments may need to adapt to *unstructured* environments (jungles or forests), or environments with few obvious features (like underground tunnels and sewers). Since we have seen no evidence of SLAM being evaluated in arbitrary environments, it is hard to predict how these algorithms will generalize to these situations.

From the perspective of a intelligent embodied agent modeling the world with objects, we predict things will go one of 2 ways: long-term mapping and localization (1) with accurate local metric representations, or (2) without accurate local metric maps. It is relatively clear how objects can be contextually represented in the case of accurate local maps (1), but we then face the following challenges: How does a robot recognize places when they are not exactly the same as the original observation? How can the map adapt over time to changes in object position and lighting? How can these local maps be effectively stored and retrieved? Long-term localization and mapping is a field in itself [9], [15], [32], [103], [110], [116], [146], [199], and while it is outside the overall scope of our current work, it is relevant to the effective functioning of future embodied systems aiming for long-term operation in many different environments.

On the other hand, it may be possible to represent objects in context without representing them in a larger metric space, as in case (2). In this case, relative spatial relationships could be stored for object instances, while the global pose of the instance itself is much less important (or unknown). This implies that object instance *recognition* would become more important than ego-motion estimation over longer time

periods, since it would be required to aid in localization. This approach is more closely related to human cognitive mapping, which focuses on significant landmarks, environment structure, and their inter-relationships over direct metric representations. Unfortunately, many existing robotic algorithms assume metric representations for planning, and combining the capabilities of topological and metric mapping to simultaneously support existing planning algorithms is a new research area.

Recognizing and handling dynamic objects and environments is particularly important for intelligent embodied systems. Unfortunately, many VO and SLAM algorithms do not support dynamic objects effectively. For example, KinectFusion and other TSDF-based algorithms rely on the stability of surfaces in the map volume to integrate enough values to represent the surface. A dynamic object moving through the map volume will therefore not create a surface. DynamicFusion[153] does handle dynamic objects, but ignores the environment in the process. It is not immediately clear how to generate and represent a static map with dynamic objects, but it will probably rely on a careful combination of the existing approaches and a flexible world representation.

2.3 Segmentation

Segmentation is a well-known preprocessing step for many algorithms in computer vision [67], [123], [189], [191], [198] and robotics [2], [34], [51], [94], [226] [6]. However, as discussed in the first section, it is exteremely difficult to get an automatic segmentation that captures what we want [16], [138]. Therefore we tend to generate more segments than objects, i.e. over-segment the scene, and then design algorithms to group these segments back together for some particular application. These segments are often called *superpixels*, since they tend to group many pixels of similar color, texture, normal and/or depth together. Superpixels are useful to help reduce the complexity of scene parsing and object recognition algorithms by considering fewer pairwise similarities or classification evaluations [145], [175].

[6] This section contains content from the author's contribution in [157].

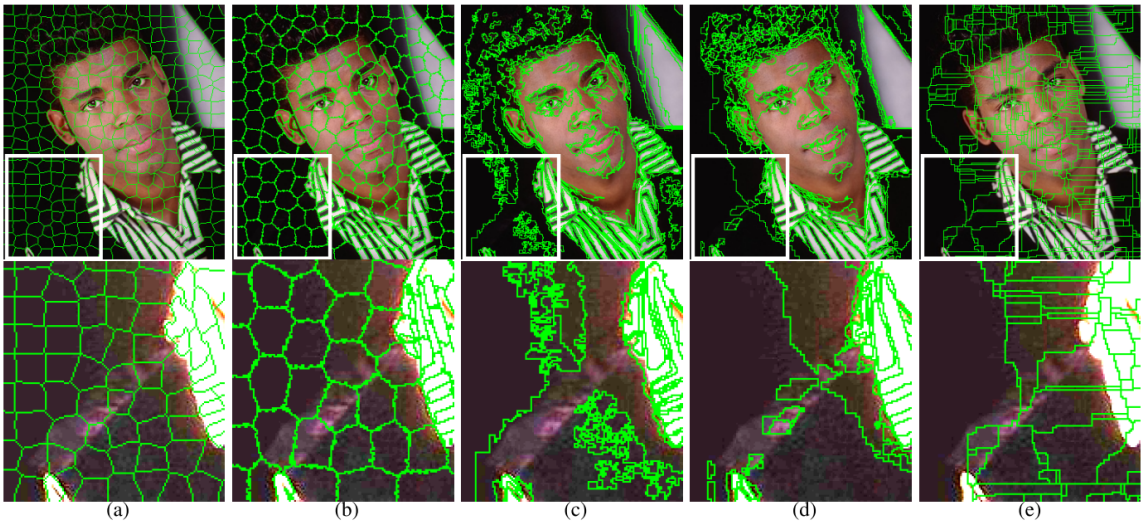


Figure 2.8 Levinshtein et al. develop the TurboPixels oversegmentation algorithm (image (a) above). TurboPixels is a form of superpixel algorithm, used to group individual pixels into larger (and easier to compute with) groups. Normalized cuts [187] output is shown in image (b). Image reproduced from [124], ©2009 IEEE.

There are multiple well-known methods that automatically compute superpixel-like segmentations (see Fig. 2.8). Normalized cuts [188], a spectral graph-based segmentation method that generates partitions that approximately maximize the similarity within a segment and minimize the similarity between segments, is a common choice for generating superpixels. The method by Felzenszwalb and Huttenlocher [55] (FH) is also used by many algorithms due to its comparitively low computational complexity. Turbopixels [124], a method due to Levinshtein et al., utilizes geometric flow from uniformly distributed seeds in order to maintain compactness and preserve boundaries. Simple linear iterative clustering (SLIC) [1] is an extension to k-means for fast superpixel segmentation. Like Turbopixels, it's initialized with uniformly distributed seeds which are then associated and updated in a manner similar to k-means clustering.

While the previous methods originally operated on images, there have been extensions to point clouds, including the algorithm we extend in this paper. Depth-adaptive superpixels (DASP) [210] is an algorithm for generating superpixels on RGB-D image pairs where each segment covers roughly the same surface area, independent of the distance from the camera. A more recent paper directly modifies the SLIC algorithm

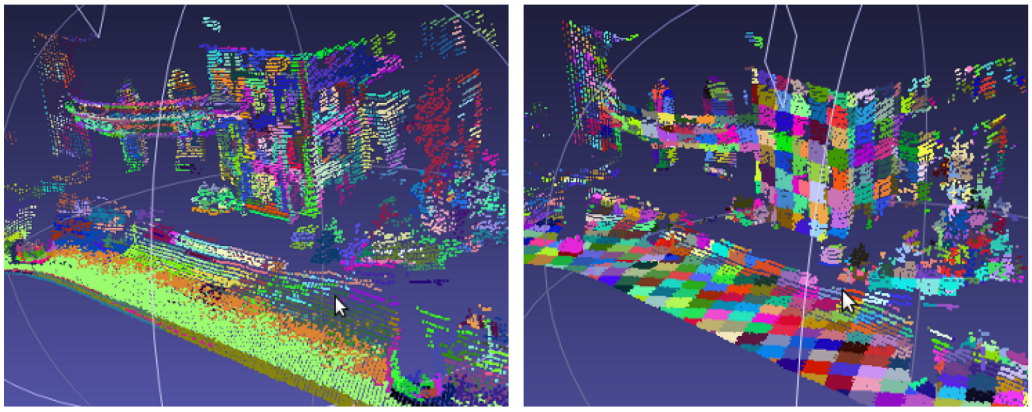


Figure 2.9 Hu et al. apply a fixed volumetric discretization to generate an efficient over-segmentation in streaming point clouds. Their approach focuses on producing usable segments for their semantic labeling algorithm. Image reproduced from [97], ©2013 IEEE.

to include depth in order to reduce undersegmentation error over depth discontinuities [147].

Incremental segment generation has primarily been applied in the video segmentation and analysis realm. Segmentation of video [25], [65], [76], [224], [225] comes in many forms, but the main focus is foreground/background segmentation, moving object tracking (often non-rigid), and multi-class semantic labeling. The common thread in these approaches is to generate segmentations that are consistent across the video frames; i.e. if a pixel belonged to one class at time T_1, then the corresponding pixel (if still visible) at time T_n belongs to the same class. While these algorithms aim for the same consistency constraints, they are only able to utilize images and therefore their techniques are necessarily more time consuming. On the other hand, they try to propagate and constrain higher level features within both static and dynamic scenes, while we explicitly take advantage of estimated camera motion and leave the semantic processing to a higher-level component.

Other research related to segmentation in streaming point clouds includes a simultaneous localization and mapping (SLAM) algorithm for outdoor multi-line laser scanners that directly incorporates segmentation and moving object detection with spatial and temporal constraints [226]. In a similar vein (although not integrated with SLAM), Hu et al. [97] apply a fixed volumetric segmentation scheme and 2.5D spatial

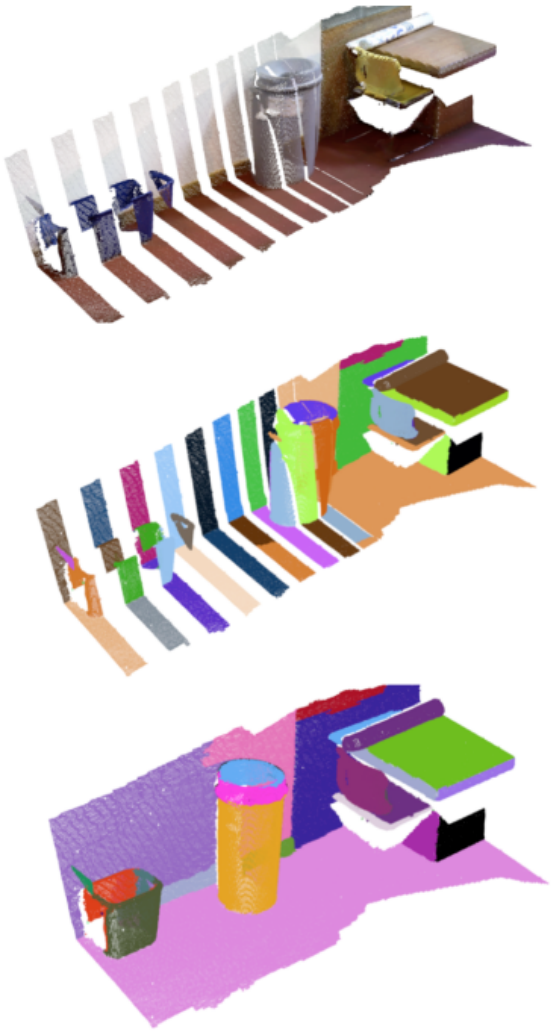


Figure 2.10 Finman et al. apply a modified version of Felzenszwalb and Huttenlocher efficient graph based segmentation to slices of a TSDF model volume, then merge the segments as additional slices are output. Image reproduced from [59], ©2014 IEEE.

data structure to aid in incremental scene classification in streaming point clouds. In that work, they are less concerned about the segmentation being proper (i.e. segments can overlap, and do not directly correspond to a distance or similarity function) and rather more concerned about efficient access and subdivision of the cloud in order to apply their scene labeling algorithm. Henry et al. [86] also apply FH segmentation to point clouds during ego-motion estimation and model building, however their goal is to use the segments as swappable and manipulable components for the purpose of generating a graph for loop closing and GPU memory management.

The work of Finman et al. [59] is closest to ours, applying a modified version

of FH segmentation to slices of a model output by the latest Kintinuous ego-motion estimation and model generation algorithm. In this case, non-overlapping segments of the world are generated by the modeling algorithm and they are segmented using the FH graph-based method. Segments are merged and recomputed as necessary according to the segment border set and the computed segment thresholds using an iterative voting scheme. The main difference in our work is our goal of consistent over-segmentation and the capability of handling overlapping clouds without a TSDF volume.

2.4 Objects and their Pose

Much of the existing work on object pose estimation focuses on using a single RGB or RGB-D frame for a known set of object **instances**. This is in contrast to generating pose estimates for a general class of objects, which may have significant intra-class appearance variation. Therefore, we divide the literature into two groups: those that focus on pose estimation for a set of object instances, and those that focus on more general pose estimation for object classes.

2.4.1 Pose for Object Instances

Pose estimation for object instances has been done for both 2D images as well as image and depth pairs (or generated 3D point clouds). While there are almost as many approaches as papers on the topic, we highlight a few salient points before summarizing some example approaches. A rather obvious observation is the prevalent use of convolutional neural networks (CNNs) in more recent work; this is to be expected given the popularity and the demonstrated power of the deep learning approach. However, this comes with a corresponding lack of flexibility as compared to a more traditional approach (e.g., 2D or 3D feature extraction and energy minimization); once trained for a specific set of instance "classes," adding another instance class is not as easy as adding a new set of features, at least without careful re-training. A slightly less obvious observation is that object-instance based pose estimation amounts to not

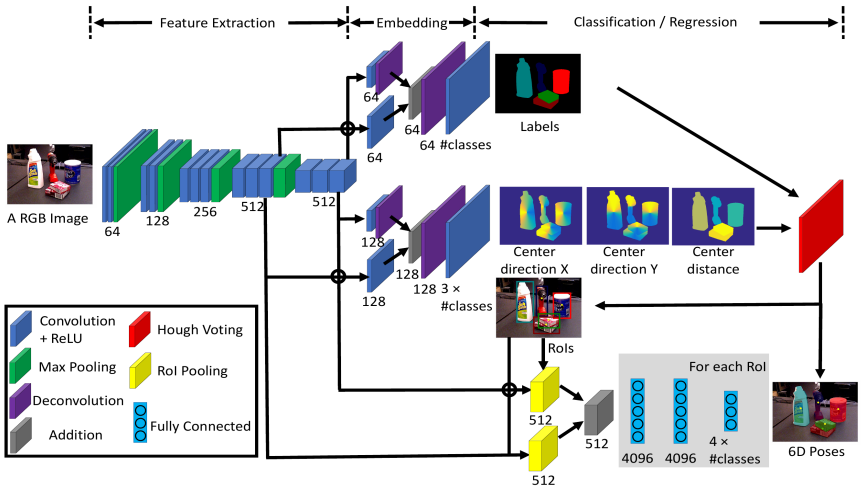


Figure 2.11 PoseCNN architecture from Xiang et al.; they train a CNN to recognize object instance pose from single RGB images. Image reproduced from [223].

much more than memorization of the instances; i.e., since each instance needs to be identified along with its pose, there is very little (if any) generalization occuring in the learning mechanism. The best performing algorithm will effectively be the best at memorization (and maybe pose interpolation, depending on the estimation paradigm). This is in contrast to some of the algorithms we discuss in the following section on pose estimation for object classes.

The pose estimation work we discuss in the remainder of this section fits in one of the following approaches: (1) single image, CNN-based regression or classification, (2) multiple image, CNN-based segmentation plus ICP, (3) single image with segmentation and RANSAC, (4) 3D geometric descriptors or dense feature correspondence and energy minimization, (5) search or reinforcement learning (rare).

Xiang et al. develop a convolutional neural network they call PoseCNN [222] for 6D instance pose estimation in cluttered scenes (see Fig. 2.11). The main contribution of the paper is a CNN architecture that decomposes the detection and pose estimation task into three main components: segmentation (pixel labeling of the object instances), translation estimation (each pixel generates a vote a direction to the centroid to regress the 2D centroid of the object in the images as well as the estimated distance from the camera), and rotation estimation where the object features regress to a quaternion representation of the rotation. The instance segmentation narrows

the input image to the rotation network to the specific region of the image where each detected object resides. These subregions are then extracted using the ROIAlign technique from Mask R-CNN [82], and fed individually to a rotation regression network that outputs a quaternion for the estimated rotation.

In related work using CNNs, Zeng et al. propose a multi-view approach to labeling and determining pose for a known set of 39 objects in a pick and place context [229]. The authors train a CNN based on VGG [24] to generate image-sized labeled segmentations of multiple views of their target scene. They obtain multiple views by using a commercial RGB-D sensor attached to the manipulator arm of the pick and place robot. The labeled segments from these views are then back projected into the scene using the depth and fused together to form a final point cloud. Models for each object are then fit to this point cloud using a slightly modified ICP algorithm to account for partial views (i.e. the model centroid is initially translated along the optical axis by half the estimated dimension to better align the surface points). In addition, they make use of the available knowledge in the hardware setup to automatically generate a large labeled dataset with minimal human intervention through automatic background subtraction and known camera and object poses.

Li et al. [126] present another CNN based method for multi-class object instance pose estimation that provides several innovations over prior work. First, pose classification *and* regression (we diagram a rotation estimation hierarchy in Fig. 8.6) are combined to improve on either approach applied in isolation; this is accomplished by selecting an almost-uniform bin in $SO(3)$ for rotation and then computing an estimated delta rotation (as a quaternion). Second, the class label is included during training by concatenating the output feature tensor with a one-hot class "image" they call a "tiled class map." Finally, they provide supervision to the feature extraction network through object segmentation. They show all three components provide improvements to estimation accuracy, while their combined classification and regression approach yields the most effective change.

The typical alternative to a CNN-based approach is learning sparse features for object instances and then minimizing some objective function over the features in order

to determine the 6-DoF object pose. Brachmann et al. [17], introduce a modification to this two-step approach by using random forests to predict both class and *object coordinates* for each pixel in an image, and then use a RANSAC-based approach to minimize a three part energy function incorporating depth, object, and coordinate penalties to determine the pose. The authors report that the approach performs better than template-based methods (specifically Linemod [90] and DTT-3D [176]) but admits some of the benefits of feature-based systems, in particular, improved robustness to occlusions.

Krull et al. explore an extension to the previous work by Brachmann et al. that employs a CNN to serve as the energy function evaluator [117]. The input to the CNN is a set of images consisting of two distinct subsets: the first subset includes the synthesized rendered view and depth image generated from the hypothesized pose (input from the RANSAC optimization method), while the second subset consists of images derived from the observed depth plus the feature images generated by the random forests. The CNN is then trained to compute an energy value representing how well the hypothesized pose matches the actual data. At inference time, the same RANSAC method (with the energy function replaced by the CNN) computes the final maximum a posteriori estimate of the object pose. Most interestingly, they show that the learned CNN is generic and is able to compute reasonable energy functions for objects that were not in the training set (although the framework does assume those object instances are known a priori and the object model is also available).

Simliar to the "analysis by synthesis" approach of Krull et al., Naryanan and Likhachev improve on the "perception by search" (PERCH) generative approach to recognizing object instance and pose in a cluttered scene [150]. As in many methods, the object model is required a priori, but in addition, the number of objects must also be supplied and the pose model is restricted to 3 dimensional space (x, y, and yaw) to make the search space more tractable. In constrast to previous PERCH approaches, this version of the algorithm handles extraneous clutter points. They use combinatorial search to construct rendered scenes of the objects, taking into considering the occlusion ordering, and then compare the rendered scene to the actual

scene. The optimization includes functions that account for labeling points as clutter, and is solved using a discrete tree search under the constraint that the objects are added in non-occluding order. While the algorithm performs well compared to other generative pose estimation schemes and supports significant occlusion and clutter for the objects, it is not clear that it is better than other discriminative algorithms, especially where efficiency is concerned. On the other hand, the algorithm does return an estimate of the object pose uncertainty, determined by incrementally evaluating the penalty on the clutter term of the optimization, which allows them to compute the certainty of the estimated poses.

Choi and Christensen take a more traditional approach to pose estimation, focusing on extracting 3D geometric descriptors to enable robust operation in unstructured environments [28] without the need to generate a segmentation before pose estimation. The authors describe the color point pair feature (CPPF), which is an extension to the point pair feature and contains the distance between two points, the angles between the point normals and the line between the points, and the colors at the points. To enable efficient feature lookup, the CPPFs must be quantized to generate a hash key. Object learning is just the memorization of all possible quantized CPPFs for an instance based on an object model mesh. The quantization parameters for the features are learned to maximize discrimination while minimizing sensitivity to noise. Object and pose detection are implemented by computing the CPPFs for the query scene, generating candidate matches and using an efficient voting scheme with agglomerative pose clustering to select the most likely pose hypotheses. The whole approach is optimized on GPU and yields detections in about 1 second. Their approach significantly improves the average precision over contemporary algorithms [42], [90], [162].

Hinterstoisser et al. also work with the point pair feature [92], attempting to improve its robustness to noise and clutter through several algorithm modifications. Sub-sampling the point cloud intelligently reduces the number of features for an object while retaining the most dicriminative point pairs, specifically keeping pairs with larger normal angles. During pose voting, better sampling of point pairs based on

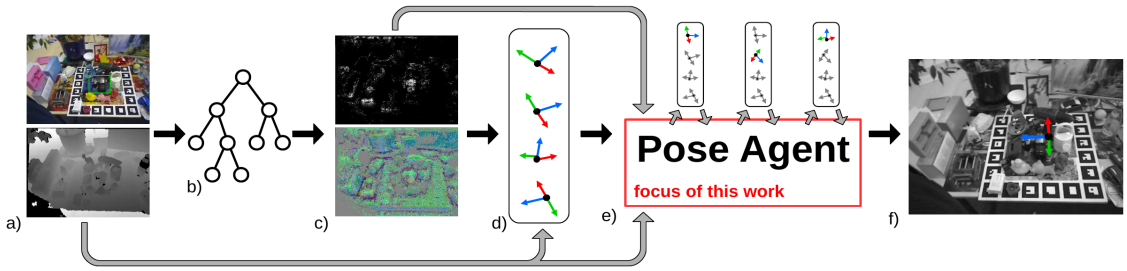


Figure 2.12 PoseAgent from Krull et al. trains a reinforcement learning agent to select pose hypotheses to refine until an acceptable pose is found. Image reproduced from [118], ©2017 IEEE.

the size of the object helps reduce noise from background clutter. Finally, spreading the distribution of quantized features into neighboring bins adds robustness to sensor noise. To avoid double counting the votes, PPFs are indexed with bit flags when they first vote. The votes are clustered in a more flexible way to account for the vote spreading, and verification is performed with synthesized depth images. While this work does not directly compare results with the CPPF work of Choi and Christensen, they do compare with Krull et al. [117] and Brachmann et al. [17], and achieve improved accuracy with lower computation times.

Do et al. present a recent approach for recovering 6D object pose [41] based on the high-performance Mask R-CNN architecture [82]. Their approach is simple; they add an additional parallel head network to estimate 6-DOF pose at the same time that the mask, bounding box, and class are being estimated. The output of the pose network is a 4 parameter vector, where 3 parameters represent an element of the Lie algebra $so(3)$ for rotations, and the fourth parameter is the translation depth; the other translation parameters are estimated using the bounding box image coordinates. They utilize a naive loss function that simply penalizes the norm of the difference between the rotation and depth parameters from ground truth rotation and depth. While they show competitive but not always superior performance to several competing systems, they do operate more efficiently based on their use of the Mask-RCNN model.

An uncommon approach to pose estimation is presented by Krull et al. in [118]. Instead of learning a specific function that maps inputs (e.g., RGB and depth) to

an output (an object class and pose), they propose to train a reinforcement learning agent to make sequential decisions about which candidate pose hypotheses to pursue for refinement within a limited budget of refinement steps. Based on the work by Krull et al. in [117], they train a CNN to learn a policy for choosing an action (selecting the next candidate to refine) given the state space. The state space contains the current available set of hypotheses, including information about how many times the hypothesis has already been refined, the original image, and the output from the random forest for every pixel. Therefore, most of the computational approach is based on their previous work, using RANSAC to select a small set of pose and object candidates. Pose refinement also proceeds as before, but is directed by the output of the specialized CNN that determines the next action through the learned policy. Learning efficiency is improved through a pre-computation step that allows the algorithm to evaluate many more policies during a single backward pass during training. The output of the CNN is a pair of energies for the set of possible actions. Note that the formulation as a sequential decision process means the algorithm can learn to recognize which candidates have a better chance of achieving the correct pose and therefore can achieve better output with less work; however, the approach as presented is rather expensive (between 17 and 34 seconds per image) even though the accuracy was improved over their previous method.

Wohlhart and Lepetit present an object classification and pose estimation approach using a CNN to learn descriptors that can be used in a simple nearest neighbor lookup [219]. The loss function for the network is based on triplets of samples where the first two elements should have a lower Euclidean descriptor distance than the first and third element. The loss function enforces a margin for dissimilar samples (whether class or pose) designed to induce larger distances for different objects but distances proportional to the difference in pose for the same object seen from a different pose. Variations in appearance due to environmental parameters are handled by a pairwise loss that takes pairwise samples of the same object with very similar pose but with different viewing conditions and penalizes non-zero descriptor distances. Training is done on both samples from the LineMOD dataset [89] as well as synthetic data. At

inference time, the CNN produces descriptors used to retrieve the nearest neighbor samples from the training set and exhibits a pose error below 20°for 96.2% of the test samples, which outperforms LineMOD and HOG-based descriptors.

In a variation on the descriptor learning approach, Balntas et al. describe a CNN-based system for learning a feature embedding for descriptors that mimic the metric distance between object poses [6]. In other words, the distance between descriptors for random image patches between two RGB-D images of an object should be similar to the metric distance between the poses of the object. The authors also recognize the challenge of determining the pose of symmetric objects, and propose a modified loss function that considers the view similarity between different poses of an object (e.g., changing the yaw of a rotationally symmetric object like a cup or bowl yields no observable depth difference) and uses this as a weight for determining the loss. Finding the best pose of a test image reduces to a nearest-neighbor lookup, where they show improved performance as compared to both Linemod [91] and the approach of Wohlhart and Lepetit [219].

While many pose estimation algorithms generate a pool of hypotheses for testing and refinement, the typical generation approach utilizes local information based on feature extraction. Michel et al. present an interesting approach for single-object pose detection that, instead, constructs a conditional random field over the features to produce a small number of globally-defined hypotheses [141]. After computing the estimated object coordinates for the image using random forests (following [117]), the algorithm constructs a fully connected graph with unary and pairwise terms over a quarter-size image. Unfortunately, a problem of that size would still take too long to solve, so the authors separate inference into 2 stages: the first solves a *sparse* non-global problem that determines the inliers to the global problem, reducing the size of the graph. In the second stage, the resulting fully-connected graph is optimized, but they further reduce the size of the problem by solving for partial optimal solutions by removing edges that exceed a thresholded distance D, which is the estimated object size. It's unclear whether the solutions are generated from anything more global than any of the other methods that restrict the search space to locally consistent feature

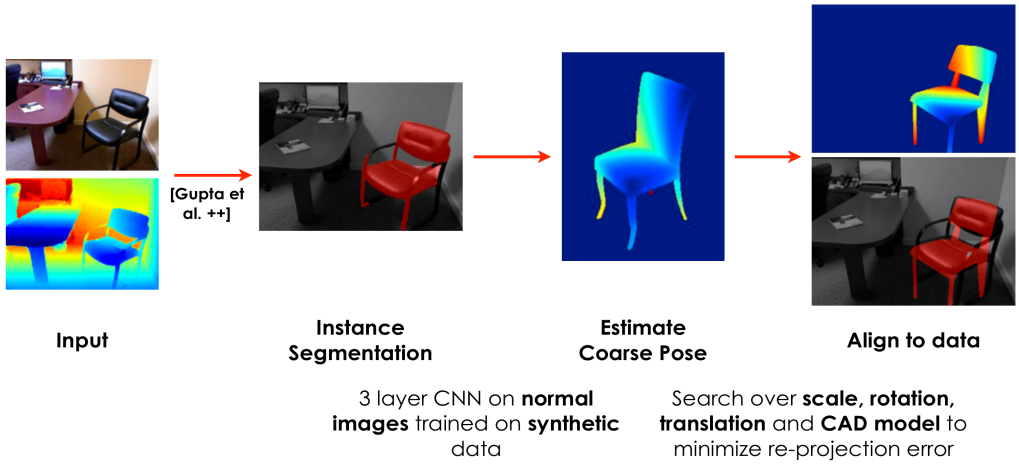


Figure 2.13 The approach of Gupta et al. first generates an instance segmentation from RGB-D images, estimates a coarse pose estimation for each instances, then searches for the best matching model to align to the instance. Image reproduced from [78].

sets, since the object will never be bigger than D anyway. While the results show it outperforming some of the authors' previous work, it is not a significant improvement over Hinterstoissers PPF work [92].

2.4.2 Pose for Object Classes

In contrast to pose estimation for object instances, estimating pose for object classes requires more generalization from the algorithm. Intuitively, this would require, at the very least, an implicit understanding of general object features, and their relation to the whole. In this section, we highlight research that de-emphasizes the focus on specific instances and attempts to recognize an object pose, even when the particular instance may not have been seen before.

Using an approach based on aligning CAD models to RGB-D scene elements, the work by Gupta et al. [78] makes the statement: "understanding such a scene requires replacing all the objects present in the scene by 3D models." I'm quite sure that is not a requirement for understanding scenes, but is certainly one way to do it, especially if there is additional semantic information attached to the matched CAD model (see Fig. 2.13). Most of the work for this paper occurs in a previous paper by the authors [79] which generates contours, hierarchical segmentations, and scene labels. They

use the output from this work to detect objects and their segments in the image. To generate pose estimates, they train a CNN on CAD models of the object classes to estimate the pose, making the assumption that the objects are supported in a canonical way (i.e. on the ground or the table) to reduce the parameter space to a single value for the object yaw. After finding the estimated floor plane and direction of gravity, they estimate the object pose and find a subset of hypothesized CAD models that match, using ICP to fit the models to the depth data.

Using a different approach, the main idea in Barnea and Ben-Shahar's work [7] is to use reflective symmetry to derive a reference frame (and partial pose) for the potential object. Using this reference frame, they build an object descriptor by collecting the surface normal values into a histogram where the normals are binned by their angle relative to the symmetry plane. This histogram is then fed to an SVN for pose classification. Their approach relies heavily on symmetry estimation, but generating reliable estimates from a single partial view is not straightforward and they do not discuss the difficulties of this task[7].

Li et al. describe a method that uses an iterative refinement approach to find an object's pose based on an initial estimate [129]. Interesting aspects of this method include the use of a deep network (FlowNetSimple [60]) for generating the pose refinements and the decoupling of the rotation and translation esimates. The algorithm is meant to be general and applicable to objects that it has not seen before. Unfortunately, this requires knowledge of the object model (or estimated model) as well as an initial estimate from another pose estimation routine. The authors do not describe how well the algorithm can converge based on the error of the initial estimates.

Finally, recent work by Grabner et al. [74] approaches the problem of pose estimation and model retrieval for object categories by first estimating the pose and then matching CAD models to the object (see the architecture in Fig. 2.14). The approach

[7]I have personally done some work on extracting reflective symmetry planes from CAD models (not even partial views) and found that finding a strong reflective plane can be quite problematic even when the full model information is present. The authors specifically show objects that have a single reflective plane (chairs) but one exemplar is shown with an incorrect plane of symmetry. They do not include objects that have multiple planes of symmetry or those with rotational symmetry. I think symmetry is an important property and has the potential to play a significant role in object and pose recognition, but this paper only just touches on the challenges inherent in the task.

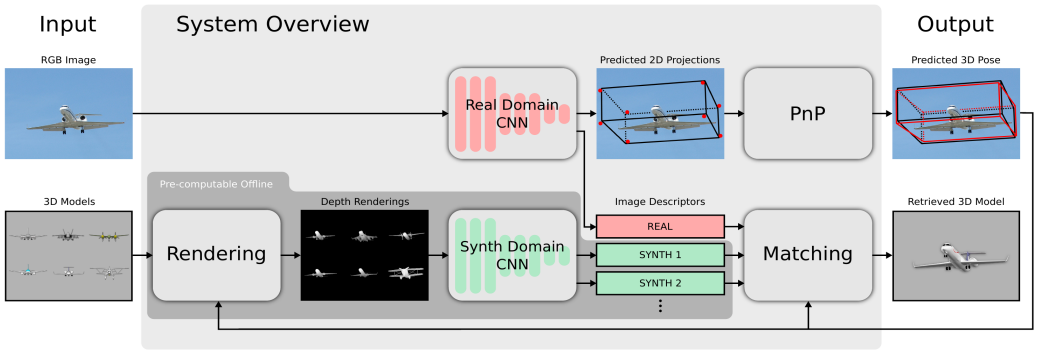


Figure 2.14 The approach of Grabner et al. estimates the 3D bounding box based on an image of an object, and then tests a single model per object category to find the best fit. Image reproduced from [74], ©2018 IEEE.

is based on virtual control points that are similar but simpler than Zhus keypoints [236]. The CNN predicts the 2D projections of a 3D oriented bounding box along with the box dimensions, allowing the algorithm to determine the pose in either a category specific or category agnostic fashion (simply by training a network on object images without distinguishing the output per category). Having the 3D keypoints for the pose estimate allows them to test a single model per class for retrieval matching. To do the retrieval efficiently, they train RGB to depth feature matches; i.e., they generate descriptors in both modalities from two different CNNs and then match the descriptors to find the best model.

2.5 Objects and their Parts

Hierarchies are an important way humans structure the world. Whole-part relationships as well as class-subclass relationships form two very important hierarchies we can use to understand relationships in the natural world as well as our "engineered" world. For example, in a company, individual employees are parts of a team, a team is a part of a department, and a department is a part of the company. Each component part often has a specific purpose or function and a corresponding name.

It is not surprising then, that we can come up with parts to describe compositional relationships in the natural world, or that we use parts to build up whole man-made objects. From a vision perspective, recognizing and understanding parts is a

useful way to describe the whole object, especially when parts move. However, even more interestingly, the concept of learning and representating object parts may help provide a basis for inferring affordances (the things you can do with an object), the intra-class variations between instances in a single class, and perhaps even functional relationships between different object classes.

In the vision community, we can call out two distinct uses of parts: semantic and explicit parts versus latent and implicit. The first usage is easily recognizable; we talk about legs, seats, arm rests, and backs as parts of chairs. We can describe their purpose and typical location, and even draw a prototypical chair using prototypical representations of the parts. On the other hand, a recognition algorithm may represent an object as a collection of latent variables laid out in space, where the variables are meant to represent the most distinctive features that allow the algorithm to recognize the object class. Where the explicit and semantic parts are easier for humans to understand and have obvious meaning, the latent parts are more for the machine; they may or may not correspond to portions of an object that we would label as a meaningful part, but instead are simply the best recognized parts according to the training data set and the nature of the features and the learning algorithm.

One might ask why the latter approach is interesting. There are two reasons: recognizing and detecting parts improves the detection of the whole object; and because learning parts automatically means there is no need for human part annotations. However, the really exciting part is when latent and implicitly detected parts provide the basis for semantic annotations (i.e., a learning system can learn about the explicit, semantic parts *without* human annotations.)

Some of the earliest part detection work started with a paper by Fischler and Elschlager [63]. They conceived of a visual detection problem within a general pictorial framework that represented the object of interest as a set of "coherent, or primitive, pieces." These parts could be arranged in a graph with spring edges between them, and used to determine whether the object is present in an image using a heuristic-based dynamic programming technique. Part representations could be pictures themselves, or more general functions that compute a value for how well the part fits at a given

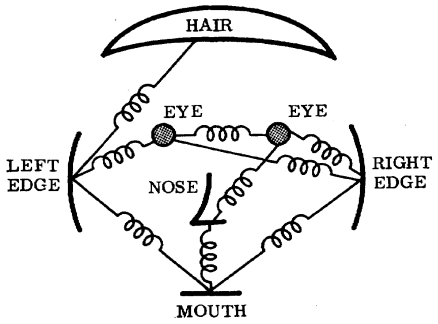


Figure 2.15 Fischler and Elschlager used "coherent, or primitive, pieces" to construct templates based on explicit and semantic parts. Image reproduced from [62], ©1973 IEEE.

location. While the parts represented the local information, the relationships between the parts defined the global structure of the object and how it should appear. The authors recognized the general utility of the approach, and showed experiments on both human face images as well as terrain recognition. This is an example of a method based on explicit and semantic parts. Specifically, facial objects such as the eyes, hair, nose and mouth are used as templates to explicitly detect the whole the face (see Fig. 2.15). In addition, the parts were not random components or portions of a face (e.g., a bag-of-words representation) but we instead semantic; when the eyes are detected, it is possible to make a meaningful statement about where the eyes were in relation to the other parts in the image.

In the 3D realm (i.e. working with a range image obtained from a laser scanning system), Navatia and Binford describe a method of recognizing an object by parts where the parts are generalized cylinders, derived from the depth discontinuities found in the range image [151]. The relationship between these generalized cylinders form the basis of the object's structural description. In this case, while the parts are explicit and (depending on the object) may correspond to semantic portions of the object, the recognition process does not start from a semantic description of the object.

In a slightly different, non-computational context, Biederman proposes a theory of human recognition that is explicitly based on recognizing objects based on parts (or components, in his words) [14]. In this framework, he describes a small set of shapes that we recognize as components of the larger object, and the way they are

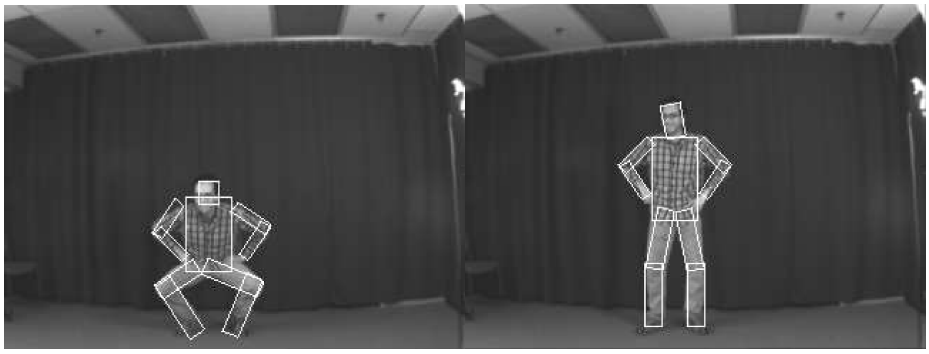


Figure 2.16 Felzenszwalb and Huttenlocher extend the framework of Fischler and Elschlager and learn the parameters for the pictorial structures model directly from data. Reprinted by permission from Springer Nature: Springer International Journal of Computer Vision, [56], ©2005.

combined helps us recognize the particular object we're looking at. While Biederman proposes no algorithmic mechanism for performing recognition by parts, the concept is obviously closely related to both of the previously discussed works. It is distinguished by having no implementation, but supports the concept through a user study that deletes portions of edges and intersections to form both recoverable and non-recoverable images of line-drawn objects. The results imply that recoverable images are those where the components are recognizable, even with degradation (e.g., through contour completion as in Gestalt psychology [108]).

In more recent work, Felzenszwalb and Huttenlocher extend the work of Fischler and Elschlager by modeling the energy minimization problem in a statistical framework [54], [56]. This allows them to learn the parameters of a pictorial structure model directly from data. They also specify a particular kind of deformation cost for efficient computation; in their case, they note that the likelihood of the part locations given the connection parameters can be modeled by a Gaussian distribution. They mirror the original face detection experiment as well as a deformable model for estimating human body pose, based on figure ground segmentation using background subtraction (see Fig. 2.16).

Another method of recognizing an object and its constituent parts was described by Cour and Shi [35]. They began with an *oversegmentation* of the image, and used an efficient search method based on a clever matching scheme that decomposed the

template distance function into distances for each segment. The overlap between each segment and the template is computed efficiently by exploiting Green's theorem, using the signed boundaries of the segment and the pre-computed integral image transform of the template. Deformable parts (i.e. parts that can move relative to each other) are supported by using the supervised template part decomposition and allowing the parts to be transformed within a small tolerance of the original junction point. Overall, given a set of labeled templates for a single class (horses from the Weizmann dataset), they can detect the objects and determine part poses within 10 ms per image.

Felzenszwalb and colleagues update the deformable part model (DPM) again to support a more general, and learnable, approach. However, the tradeoff in the new approach is that parts are no longer explicit and semantic [52], [53]. Instead, parts are latent models to be learned while being exposed to objects within a class; however, with this tradeoff comes the *benefit* of no longer needing to produce labeled parts. In this case, a single part model (represented as a learned histogram of oriented gradients (HOG) template) *may* correspond to a semantic part, but the model itself cannot associate labels to the parts. The algorithm is quite robust and was state of the art (until deep learning and convolutional neural networks took over); the user specifies the number of parts, and the system is carefully trained to find the best discriminating part models as well as the deformability parameters (modeled as Gaussians) to detect the target objects. While the deformability of the parts (based on a simple star graph from a root part to the peripheral parts) handles some variations in the object shape, the authors also describe a method to handle multiple poses as a collection of models. Many variations on the approach have been proposed, including work in 3D [57], [166], as well as work that shows DPMs can be interpreted as CNNs [73].

What if it were possible to learn common object parts without explicit human labels. This would be one step along the road to Biederman's theory of recognition by components, and could aid autonomous systems with not only recognition tasks, but also grasping and reasoning about affordances. Work by Tulsiani et al. takes this first step using 3D object models from ShapeNet and learns to generate consistent

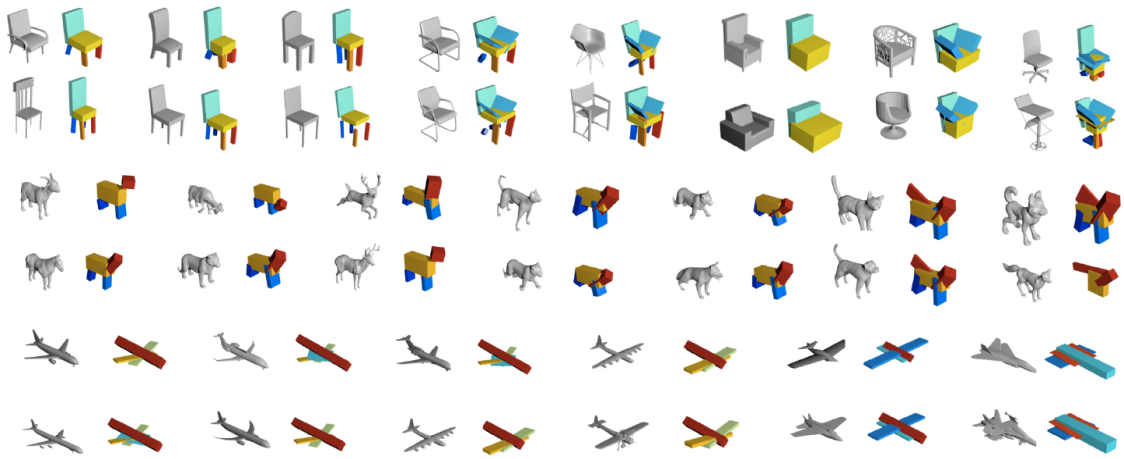


Figure 2.17 Example part primitives generated using the volumetric primitives approach from Tulsiani et al. Reproduced from [203].

primitive components for several object classes [203]. In their approach, objects in a canonical pose are discretized into a volumetric representation and provided as input to a 3D CNN that outputs a regressed set of parameters defining up to k cuboid primitives (i.e., 3 parameters for the cube scale along with 7 parameters describing the cube's transform to the part position). In order to handle variable numbers of parts for intraclass variation, they additionally estimate an existence probabilty p_k for each part; during training, they minimize the average loss of a set of samples over the induced shape distribution using the REINFORCE algorithm, adding in a small reward for parsimony to help simplify the resulting shapes (see Fig. 2.17 for example output).

In related work, Yi et al. learn a hierarchical part labeling from online shape repositories by taking advantage of weakly supervised data [228]. In this case, they split the task into two parts: (1) extracting hierarchical structure and labels common to human models (by exploiting the geometry commonalities and the human-generated labels embedded in the computer aided design (CAD) models), and (2) using a Markov random field formulation over the mesh to classify the parts according to the previously extracted data.

Very recently, Mo et al. presented PartNet [144], a benchmark dataset for fine-

grained and hierarchical part-level annotations[8]. In this massive work, the authors use expert-defined hierarchies to guide professional annotators in generating fine-grained segmentations for over 26,000 objects in 24 categories. They also propose a part instance segmentation algorithm based on the PointNet++ algorithm [172], extended to output semantic labels for each point, estimated part instance masks, and a confidence for each instance. Their algorithm outperforms a state-of-the-art point cloud segmentation algorithm trained using the PartNet dataset.

[8]In chapter 8, I describe my version of PartNet, which is a CNN designed to determine object pose using part instance segmentation. This was developed concurrently with the PartNet described in [144]; I found out about the work the very night of my dissertation defense!

Chapter 3

Sensor Modalities and Data

Like many in the computer vision community, I would prefer to have a robot operate with as few as two visual sensors, i.e., a stereo camera pair, mounted in whatever serves as a head for the robot. This is attractive not only because it mimics nature, but because it is simple and low-cost (from a hardware point of view). Having two cameras allows stereo vision algorithms to compute the depth of the environment where needed (most of the time), and cameras are passive devices that do not emit radiation. This latter attribute not only reduces energy use, but makes it harder to detect robots that may be used in sensitive situations (e.g., certain military, rescue, or sensitive socio-political situations).

However, *unlike* many in the computer vision community, I also see the utility in taking advantage of sensors that may give me more information to work with. While I hope we eventually get to the point that binocular stereo cameras are sufficient (perhaps with the capability of vergence), using 2D and 3D laser scanners can give a robot enough accurate information about the environment that it can make good decisions without hedging too many bets. The purpose of this chapter is to introduce the sensor modalities often used for perception tasks on robots, and talk about some of the fundamental ways we can capture data and perform computations on them. We'll look at sensors with increasing complexity, starting with a 2D laser line scanner, moving to the pinhole camera model, and finally exploring 3D sensor configurations, along with their benefits and drawbacks.

Figure 3.1 A Hokuyo UTM-30 scanning laser rangefinder [93]. This sensor is quite popular on many mobile robots.

3.1 2D Laser

Most 2D laser scanners (also called LIDAR sensors) in use on robots operate under the principle of time-of-flight, i.e., they measure the time it takes for a laser pulse to return from a distant object, and then compute the distance using the recorded time and the speed of light. Since the laser beam is focused coherent light, it is quite accurate and can estimate distances with low error, often less than 0.05% of the distance measured. A 2D laser scanner rotates a laser beam in a plane, taking many distance measurements over some active angular range covering half to three-quarters of the circle.

One or more 2D laser scanners are often used for obstacle detection and avoidance, as they return accurate distances to objects surrounding a robot at wheel level at high rate (a popular sensor, the Hokuyo UTM-30 and its variants (see Fig. 3.1), scans a 270 degree portion of the circle at a rate of 40 Hz). Robots can use these sensors with pre-generated maps of the building structure to localize (find out where they are) themselves with a high degree of confidence. These sensors are often used for mapping the local environment, or even a whole building floor using algorithms such as simultaneous localization and mapping.

The immediate benefits of adding a 2D laser scanner to a robot are large: there are off-the-shelf algorithms for localization, planning and obstacle avoidance in office-like environments that directly use the output of these sensors. Many 2D laser sensors do not take up much space, and can even be used on medium-sized aerial vehicles such

as quadrotors for more advanced mapping tasks.

However, typical 2D scanners used on robots today are not inexpensive: the model mentioned earlier is around $5,000. There are cheaper models available with lesser capability, that may be sufficient depending on the requirements of the task. Lower angular resolution, slower scan speed, and shorter ranging distance are among the attributes you may trade off for lower cost hardware. That said, these sensors are pretty standard inventory for most commercial and research-grade mobile robots you can find on the market today.

3.2 Camera

Just about everyone is familiar with the operation and output of a typical digital camera these days. In short, a camera produces a 2D image of a 3D world by recording the projection of the light rays in the environment onto some kind of imaging surface. The implications of this (which we discuss later) are important when we are trying to understand the actual 3D structure of the world.

There are many ways to interpret the definition of an image. While an image may be generated by a variety of mechanism, most can be categorized as either capturing or rendering. Our own eyes capture images of the environment and cameras capture images. From a robotics and computer vision point of view, we are mostly interested in captured images of the environment. Therefore, we conceptually define an image to be the projection of the light emanating from some volume of 3D space, through a single focal point, onto a 2D plane from some point of view. This produces a **perspective** projection of the 3D world. Given an object with fixed size, its image appears larger the closer it is to the imaging device, and smaller the farther away it is.

While film cameras have been around for a while, digital cameras have only been around since 1975 [171] (perhaps earlier than you may have expected). Digital cameras collect light through lenses just like any other camera, but instead of exposing film with the light, an array of sensors is exposed with light. In this case, each picture

element (i.e., pixel) records a value proportional to the amount of energy falling at that location. The pixels are arranged in a 2D grid, and the sensor records the values from the pixels either line by line (i.e., a rolling shutter) or all at once (i.e., global shutter). This implementation detail affects the quality and content of the images received from the camera, since the motion of the robot and the motion of dynamics objects in the scene yield different results with different sensors. Most importantly, rolling shutter cameras produce distorted images of moving objects, and therefore global shutter cameras are the preferred choice for machine vision and robotics.

Digital cameras can provide a 2D image of the 3D environment directly to a perception algorithm. However, it is still not (usually) possible to use that image immediately due to other distortions that may be present. Like film cameras, digital cameras still use one or more lenses to focus the incoming scene onto the imaging sensor. These lenses can cause distortion of varying kinds that hinder our use of cameras in perceiving the true geometry of the environment. For example, a lens may produce a barrel effect (many wide angle lenses do this), or may introduce other decentering effects based on the shape and manufacturing process of the lens. For standard perspective lenses (the discussion of very wide angle "fisheye" lenses and processing is outside the scope of this section), we are really aiming to reproduce the pinhole camera model. This model is an idealized version of a camera that provides straightforward geometric relationships that govern how the 3D scene is represented on the image plane. Figure 3.2 illustrates the model visually.

The full relationship between a world point and an image point is governed by both the intrinsic and extrinsic parameters of the camera. The intrinisic parameters include the focal length f (shown in the figure), the coordinates (c_x, c_y) of the projection of the optical center O on the image plane, and the coefficients $(a_1 \dots a_5)$ for the radial and tangential distortion models (due to Brown and Conrady [19], [20], [33]):

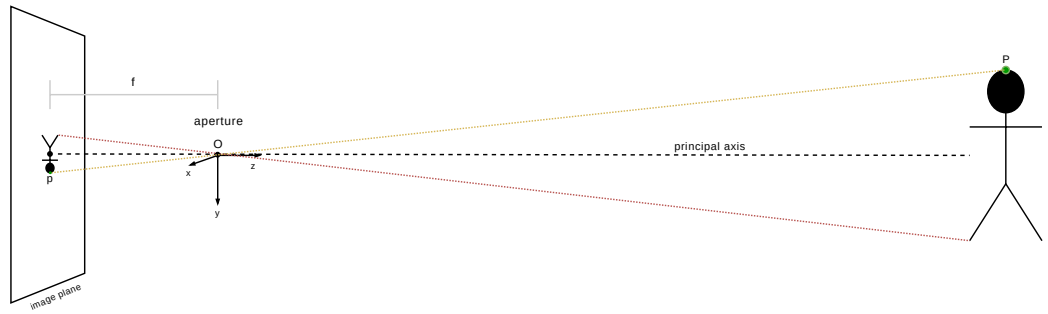


Figure 3.2 A simple pinhole camera model. An image in the real world is projected through the camera's aperture (i.e., the pinhole) onto the image plane. The relationship between world point P and the image point p is described by the projection function π described in the text.

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \underbrace{(1 + a_1 r^2 + a_2 r^4 + a_3 r^6) \begin{bmatrix} x_n \\ y_n \end{bmatrix}}_{\text{radial part}} + \underbrace{\begin{bmatrix} 2a_4 x_n y_n + a_5(r^2 + 2x_n^2) \\ a_4(r^2 + 2y_n^2) + 2a_5 x_n y_n \end{bmatrix}}_{\text{tangential part}} = f_{\text{dist}} \left(\begin{bmatrix} x_n \\ y_n \end{bmatrix} \right) \tag{3.1}$$

The full image generation equation is then:

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = f_{\text{dist}} \left(\pi(KTP) \right) \tag{3.2}$$

where π is the projection function that divides the point **KTP** by z. **K** is the intrinsic camera matrix, and **T** is the extrinsic transform of the camera with respect to the world. **K** is defined as:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

and **T** is:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R}_c & \mathbf{t}_c \end{bmatrix} \tag{3.4}$$

where $\mathbf{R}_c \in SO(3)$ and $\mathbf{t}_c \in \mathbb{R}^3$. Since $\mathbf{T}$ is a 3×4 matrix, we use homogenized world points $\mathbf{P} = (X, Y, Z, 1)$. f_x and f_y are really the product of the pixel scale values and the focal length: $f_x = s_x f$ and $f_y = s_y f$ (in case a sensor has a non-square pixels).

3.2.1 The consquences of projection

Let us consider the consequence of turning the 3D world into a 2D image. We note the obvious: we lose a dimension! Interestingly, the aspects of the environment we no longer perceive are dependent on the pose of the camera in the world, which leads us to wonder if we can obtain the missing information by moving the camera around. The answer is yes, up to a point; the algorithm to do so is called "structure from motion," and uses multiple poses of a camera viewing a scene to reconstruct (some of) the structure within the scene. The reconstruction has one drawback, howeer, and that's the fact that we cannot know the scale of the scene. How does one "unit" in the reconstruction relate to a meter (for example) in the real world? It is not possible to know this without additional information: if you recall the diagram of the camera model and perspective projection function, it is easy to see that anything along the line of the projection from a point in the world to the image plane yields only the thing closest to the camera. This is not too surprising; we're familiar with this concept as our own human vision system generates images of the world in a very similar way (with a single eye replacing the camera construct). However, what may not be completely obvious is that the bit of the world we see at one point may also occur at any point along the projection ray. This is where the scale ambiguity comes from. Humans use the vergence of our binocular vision system to estimate depth and we also use known relative sizes, foreground/background motion, and possibly other cues to infer depth and structure, even without binocular vision (this is how people with only one eye can still successfully drive vehicles).

Unfortunately, this extra information is derived from the semantics of the image, and not inherent in the image information itself. Knowing how the camera moves in the real world could provide scale, and that is the approach some visual odometry algorithms exploit when cameras are paired with an IMU to do visual-inertial navigation. However, would it be more useful if the robot could just sense 3D in the first place?

3.3 3D

Seeing the world "as it is," that is, veridically, is one of the reasons 3D sensors are so popular for robots. The purpose behind a 3D sensor is to produce measurements of the world observed from the coordinate frame of the camera that correspond to distances we, as humans, can directly measure in the world. Given enough measurements from the sensor and knowledge of how the sensor moved within a static environment, we could generate a 3 dimensional reconstruction of the environment in the memory of a computer. This would be useful for a variety of robot tasks, including grasping and manipulating recognized objects within the world, as well as navigation, motion planning, and place recognition. This is the primary reason you may see the expensive 3D laser scanners on the roofs of self-driving vehicles: the scanners help provide (mostly) unambiguous metric measurements of the world at a given time instant, and these help the car localize itself within its known map as well as avoid obstacles that may not have been recognized by the camera sensors.

We'll talk about 3 sensor modalities for 3D in this section, two of which are image based, while the third is point cloud based.

3.3.1 Stereo

A stereo vision system uses 2 or more extrinsically calibrated monocular (*single eye*) cameras to enable the computation of depth for certain portions of their simultaneously captured images. Stereo systems do not directly observe the 3D point since they don't make direct measurements the same way a time-of-flight based laser sensor does.

Instead, pixels are matched between the left and right images in order to compute the disparity. Disparity refers to how far apart the matches pixels are in image space. Since the cameras are viewing the scene from slightly different positions, a single point in the world will be observed at slightly different locations on each camera's imager. Objects that are far from the camera will have smaller disparity, while objects closer to the camera will have larger disparity. When we compute the extrinsic calibration between the cameras in a stereo pair (i.e., determining the metric distance and rotation between the camera views), we are able to use the metric distance to triangulate the point in space. For a more technical discussion of this process, see the classic on multi-view geometry, Hartley and Zisserman [81].

One of the primary challenges with stereo vision is finding the correspondences between the pixels in the two images in order to compute the depth. Without *correct* correspondences, you will get an incorrect depth value. This means reliable depth computation using stereo cameras relies on distinguishing features in the scene and minimal repeating structure. Repeating structures can cause problems by generating matching in multiple positions along the epipolar line, making it difficult to determine the correct disparity. Some algorithms use special techniques and heuristics to generate dense depth output [70], while other algorithms simply compute sparse points based on strongly matching indirect features.

3.3.2 RGB-D

RGB-D sensors operate on a principle similar to stereo but *can* perceive depth even when there are no observable features in the scene. How is this accomplished? For RGB-D sensors based on structured light stereo, the sensor effectively projects a texture onto the surface and uses what it knows about the texture to locate known features. Then, observing how those features interact with the world relative to a "canonical" view of the texture allows the algorithm to derive a disparity value. If we project a known texture onto a wall from a fixed position relative to the observing camera, then we can compute where the known features are observed in the camera and find the disparity from which they were projected. As in a binocular stereo

system, we can use this disparity to compute depth.

Structured light stereo systems consist of a projector and a receiving camera. The projector illuminates the environment with a texture image, often in the infrared range, so it's not observable to the naked eye. The receiving camera observes the texture and its interaction with the world and uses it to compute depth with known patterns within the texture. These two devices are often paired with a second color camera that is extrinsically calibrated with the other camera, and the system together can produce RGB-D images. A frame from an RGB-D sensor is actually a pair of images, one color and the other depth, registered so that that pixel 42 in the color image corresponds to pixel 42 in the depth image.

One of the first commercially available sensors of this kind was the Microsoft Kinect. This device was originally developed as a controller or input system for the Microsoft Xbox video game console, but was immediately recognized as an affordable depth sensor by hobbyists as well as vision and robotics researchers. Because of its onboard depth map computation at frame rate, you get a depth image and a color image computed automatically by the hardware at 30 frames per second. This produces a lot of information very quickly; the valid pixels in the depth map are almost as dense as the resolution of the camera images directly, which yields denser depth maps than most stereo camera pairs of the same resolution. Transforming the depth map into a point cloud usually yielded around 262,000 or more points *at frame rate*.

RGB-D sensors are still very popular, but they suffered from two drawbacks. First, unlike regular monocular cameras, and unlike standard stereo vision systems, they requires an active source of energy to illuminate the world with an artificial pattern in order to perceive depth. This means that you're limited in the depth you can perceive both due to camera resolution but *also* because you must be able to discern the texture. This is analogous to the problem you have likely experienced in real life, when you are shining a relatively bright flashlight down the road at night, and the effect of the light falls off with greater distance. The second problem is directly related to the first, but probably more problematic; you cannot use most RGB-D

sensors in the daytime because the sun's energy saturates any information you might get back from the structured light projector.

Finally, there is another way of generating RGB-D data: pairing a 2D color camera generating RGB information with another 3D sensor (such as a laser scanner) that does not observe color information. With the correct extrinsic calibration between the sensors, it is possible to "paint" the resulting point cloud from the 3D sensor with the colors from one or more color cameras. We demonstrate this in our later chapter on multi-sensor graph calibration (see Fig. 4.1 in Chapter 4).

3.3.3 Laser

Finally, just like scanning laser range finders can directly measure depth in a single three dimensional plane of the world, rotating the plane over time will cause the laser to scan the world outside of the plane. Therefore, when we do this quickly (perhaps many times a second) we can generate 3D point clouds. A common way of doing this is to scan a vertical array of lasers around a vertical axis. Each laser maps out a (truncated) cone in the world and generates depth measurements at some point along the surface of that cone. When you put them all together, you get a 3D point cloud of the world. This point cloud may be less dense in the vertical direction than a stereo camera or an RGB-D camera depth based point cloud, but in many cases, it has features that counter those drawbacks. For example, a popular sensor used in many self driving vehicles and other robotic systems is the Velodyne [206] 3D laser scanner. A sensor of this type may have up to 128 individual laser units arranged in a vertical array that are rotated very quickly; this captures a point cloud around 10 times a second. While the individual scanlines are dense, the overall point cloud can be sparse depending on the distance to objects and the motion of the scanner. This makes it challenging to build up models of the environment from single scans. However, the speed at which the system rotates is fast enough to effectively see 3D dynamic objects in real time. This is especially useful for obstacle avoidance and for high resolution localization. Given the accuracy of the LIDAR process, the point clouds from laser scanner such as these can be used for very accurate localization and

mapping (especially when paired with cameras) as in the work of Zhang et al. [231].

3.4 Representing surfaces

Depth images from RGB-D cameras (as discussed in section 3.3.2) are also known as range images. A range image is an image where the values represent the depth from the camera origin to the surface in the scene. Instead of recording intensity, the image records a distance. However, due to the nature of the sensing device, it is common to have unknown values present in the image (often indicated by 0, ∞ or `NaN`), where no depth was measured or no surface was present up to the measuring limit of the sensor. Range images are one form of data representing the 3D structure of the world from a single point of view.

In contrast to an image containing projected 2D intensities, however, we know the value z and given the intrinsic parameters of the device that generated the range image, we can compute the world coordinates of the point P, up to the error in the calibration and uncertainty of the range estimation. This allows us to construct an **organized** point cloud using the range image data, such that each point is relative to the pose (or frame) of the sensor and corresponds to a pixel in the range image. Knowing z we solve for P_x, P_y with the following functions:

$$x = \frac{z(u - c_x)}{f} \qquad\qquad y = \frac{z(v - c_y)}{f} \tag{3.5}$$

A point cloud that is **unorganized** does not have an image-based organization to the points. In other words, it is not possible to look up point (i, j), and then trivially find the view-dependent neighbors $\{(i+r, j+c)|r, c \in \{-1, 1\}\}$. This has implications in both the efficiency of algorithms that use neighbors as well as preventing algorithms that need to make use of the view-dependent structure of a organized point cloud.

The common methods of determining neighbors in an unorganized point cloud depend on the construction of spatial data structures. These are data structures that

store points in a structured way so as to make a neighborhood search more efficient than a linear scan (i.e., calculating a distance function to every point in the dataset). They accomplish this by analyzing the distribution of the input data along spatial axes and subsequently partitioning space in some way to reduce the comparisons required at each step by some factor (the usual aim is to eliminate as close to half of the remaining points as possible). For example, a binary space partitioning tree splits the volume at every internal node, such that at the first node, going left or right would eliminate half the points in the volume that lie on the right or left, respectively. A k-d tree (or k-dimensional tree) is a special case of binary space partitioning tree, where every internal node splits the data set using a single dimension that best partitions the volume. The complexity of a spatial data structure such is $O(n \log n)$, since, like a sorted list, we must make $\log n$ steps to add a new datum to the tree, for all n data.

The loss of view-dependent structure brings another challenge: it's not easy to determine depth discontinuities. These are often an aid to segmentation algorithms, and in some cases are used to help determine features. Even if one chose a viewpoint pose within the space of an unorganized cloud, calculating the surface discontinuities would still involve calculating the surfaces, another unsolved problem with a large body of research for a solution!

To distinguish completely unorganized clouds from their triangulation- or graph-based counterparts, we will call the latter a "surface-based" point cloud. While this nomenclature is not the most evocative, it highlights the fact that view-based neighbor lookup is not a simple indexing operation, and that we also have semantic information regarding estimates of point connectivity for determining surface properties. In general, there is no guaranteed maximum degree for each node in the point graph (even triangulations may have triangle-fan structures). Not all surface-based point clouds are graphs; some may be represented by lists of points with triangle indices, or slightly more advanced representations using data structures such as half edges (which help encode topological relationships between points, edges, and faces). Since we have not (yet) created a sensor that can directly capture surface definitions from the environment, we must find some way to construct surface-based representations from point

clouds. Surface inference is the process of recognizing surface properties from the samples gathered from one or more observations of the surface. These properties can include finding the bounds of the surface in the environment, the analytical form of the surface for a given bounded region, as well as textures in both the geometric and intensity domains. Knowing surface properties may help to:

- interpret the information in a sampled 3D scene,
- reduce or remove noise from the sampled representation,
- compress the sampled data (i.e. remove the samples) by replacing regions of the scene with the surface properties,
- aid the grasping task by relating sampled data to known, graspable shapes,
- support the object recognition task by recognizing surface parts,
- and contribute to semantic definitions of the environment geometry (find walls, floors, ceilings, horizontal and vertical support surfaces like chair backs and bottoms).

The challenge is to take noisy, contradictory, non-uniform, and incomplete sets of surface samples from multiple viewpoints and turn them into coherent, useful representations of surfaces. While detailed descriptions are outside the scope of this chapter, we highlight a couple algorithms of interest that provide food for thought when considering the larger problem of visual understanding.

The first algorithm does not explicitly calculate a surface; instead, it makes the assumption that points in a point cloud are noisy samples from a surface, and proposes a family of likelihood maps that represent the probability of a surface passing through a given point. Pauly et al. [165] compute surface fitting estimates (in this case, weighted least squares plane estimation) for every point $\mathbf{p}_i$, and then use these to calculate the likelihood that some point $\mathbf{x}$ has a surface passing through it and all nearby points. The second algorithm from Kolluri et al. [109] computes a Delaunay tetrahedralization of the cloud and then performs a classification of the tetrahedrons

into “inside” and “outside” volumes, such that the boundary between the volumes determines the surface of the cloud. The technique utilizes spectral partitioning to help disambiguate outlier points from surface points, based on the method of normalized cuts [187] (slightly modified to consider negative weights in their **pole matrix**, the altered version of the graph Laplacian).

Chapter 4

Extrinsic Multi-sensor Pose Calibration

4.1 Introduction

Robots need to perceive the world in order to accomplish just about any task worth performing; why have a robot if not to interact with the world? In this dissertation, we focus almost entirely on the perception part of the interaction process (in contrast with the manipulation part). Perception is almost always a necessary prerequisite before deliberation and action: the robot must interpret the state of the environment to determine how it has changed since the last "look," what objects and obstacles are present, how the environment is configured, where it is in relation to any goal locations, and how the environment state relates to any current ongoing tasks. Some of these interpretations overlap, but the underlying principle is clear: the more information the robot has about the world, the better it can perform its assigned tasks.

For example, Table 4.1 lists some tasks a robot might perform in a *military* work setting. This list is not at all exhaustive, but gives an idea of the variety of tasks a military robot could perform. In some cases, specialized hardware may be required, but in all cases, the robot must be aware of and understand important features of the environment.

Table 4.1 Examples of military work tasks for robots.

Logistics	Troop Support	Reconnaissance
material transport	carry heavy equipment	area overwatch
troop transport	carry wounded troops	map and search
guard duty	support forward/rear surveillance	enemy observation
resupply	IED detection	change detection

This is reasonable and intuitive. However, the details of *how* it accomplishes this understanding, are harder to enumerate. In addition, robots are not humans. We can mount a variety of sensors on a robot, and expect to use different sensors for different tasks, or groups of sensors for any single task. We have previously discussed useful, readily available sensor modalities. Let's imagine that we mount these sensors on an imaginary robot in a particular configuration. Many tasks may benefit from fusing different sensor modalities; all perception tasks benefit from knowing the poses of the sensors relative to the robot frame, and their poses relative to each other. For example, say we mount a Velodyne 3D laser scanner and multiple color cameras on the robot. If we know where the cameras are in relation to the Velodyne, then we can project the laser points into the camera frames, and determine the color of the points in the real world.

The question is: how do we know where the sensors are? Even if you have all the engineering drawings for the sensors and the mounting hardware, it is not possible to determine where the actual sensors are without some kind of extrinsic sensor pose calibration process. This chapter discusses the algorithm we developed to compute the relative sensor poses for multiple sensor modalities mounted rigidly on a robot structure. In many ways this project began out of necessity: we had several sensors on a robot we use for a variety of tasks and no way to determine their poses outside of playing guessing games (suprisingly enough, this is a very commonly used tactic)!

Calibrating sensors (both intrinsically and extrinsically) is a tough and under-appreciated problem. Most researchers who work with robots are familiar with the requisite yet error-prone process of determining the poses of multiple sensors in order to fuse the sensor data into a single reference frame. Most of the software used for

calibration focuses on sensor intrinsics, and in some cases only pairs of sensors. What do you do if you're lucky enough to have a robot with more than two sensors, let alone sensors with multiple modalities? In addition, current systems require careful steps, and even some significant manual involvement. We would like an algorithm and system that minimizes direct involvement from the user, handles as many sensors *and* sensor types as possible, and makes it easy enough to collect data that it won't take all day.

Therefore, we aim to create a system in which a single data collection can automatically produce a globally optimized calibration of an (almost) arbitrary sensor configuration. In this work we describe a method and system that computes the relative poses for multiple environment sensors with differing modalities and varied acquisition rates using graph optimization[1].

Sensor fusion research in robotics would benefit from a calibration procedure that is run interactively and online using only the capabilities of a mobile robot and its current environment. While an algorithm that does *not* require a calibration object would be ideal, the data association problem makes this difficult to achieve, which is made even more challenging by differing sensing modalities. Therefore, we constrain the data association problem by making use of a custom calibration object, shown in Fig. 4.4a, with the expectation that using the object will facilitate accurate calibrations.

The system we present makes few assumptions about the number, relative pose or fields of view of the sensors; in fact, the only requirement is that any single sensor has a partially overlapping field of view with at least one other sensor, so that the calibration object is always observed by at least one pair of sensors at the same time[2]. The sensor type determines the representation of the object that is detected and stored at each distinct pose of the target. We assume that the calibration target is planar, which allows us to construct the geometric relationships between individual sensor detections required for calibration. Finally, we assume that each sensor has

[1]Portions of this chapter are adapted from [158] ©2015 IEEE.

[2]This is why we say the algorithm can handle an "almost arbitrary" configuration of sensors.

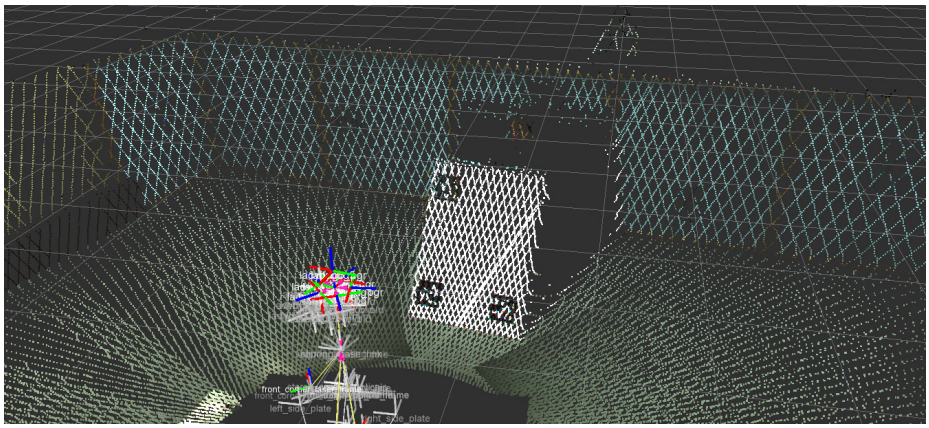


Figure 4.1 An example camera-colored 3D LRF frame from the calibration data collection.

been intrinsically calibrated, and do not include intrinsic parameters as part of the problem formulation.

The full calibration procedure can be divided into four phases: data collection, target detection, pairwise calibration, and graph-based global refinement. In short, the data collection phase involves walking around the robot with the calibration object, and using a remote trigger to indicate a frame capture. After the sensor data is recorded, calibration object detections are performed for each sensor modality, and the detections are associated into sensor pairs for the next phase. These detections form a graph with a single connected component of sensors, where an edge represents a sufficient number of co-occurring detections between a pair of sensors.

Given the data collected for each pair, the pairwise calibration phase proceeds by coalescing all sensor detections for each pair into a single dataset. We then determine the relative pose beteen each pair using an appropriate optimization routine embedded in a RANSAC framework to automatically filter outliers.

Finally, these initial pairwise estimates are used to construct a hypergraph that contains the sensors *and* calibration object detections as nodes as well as several types of edges representing the geometric constraints between the sensor modalities. We use non-linear optimization to minimize the error over the entire graph using the g2o framework [119].

Through both simulated quantitative results and real-world qualitative results, we

show that the resulting framework successfully calibrates a number of different types of sensors with minimal operator intervention, and the global optimization is shown to be more accurate than the initial pairwise calibration.

4.1.1 Multiple modalities and other complications

Attempting to calibrate sensors with differing modalities is difficult since they produce different kinds of data. Cameras generate 2D images, projecting what they see in Euclidean three-space onto an image plane. 2D laser scanners generate an in-plane scan of ranges from the laser origin, and 3D laser scanners generate point clouds (often without additional information; we've found that the provided intensity information is often not dense or reliable enough). Each sensor produces a different "view" of the world. In order to determine the relative transform between any two sensors, each sensor must be able to see some known object or set of features (let us conveniently call it a calibration target) simultaneously and be able to reliably determine the pose of the target.

All sensor modalities must be able to generate recognizable, measurable, and preferably accurate observations of the target. Given the typical set of modalities we observe on robots: 2D lasers, 3D lasers and other 3D cloud generating sensors (e.g. ToF ranging, structured light stereo), and cameras, we need a calibration target that can be reliably measured in a coordinate frame relative to the sensor as easily as possible. Finally, the object must be distinguishable from the surrounding environment, in order to be able to associate the different observation modalities with each other.

Depending on the nature of the target, we must observe it in multiple poses to better constrain the relative transforms between pairs of sensors. For example, imagine that two 3D sensors observe a planar calibration object from two different viewpoints. Assume the sensors can reliably estimate the plane parameters for the target (we interpret the observation as evidence of an imaginary infinite plane, this helps to account for partial observations of the target, and can increase the amount of evidence we can use to estimate a transform between the sensors). However, if

the target only moves in the plane it represents (admittedly a low probability event), then we only constrain 3 degrees of freedom, leaving 3 degrees unconstrained. While one might make use of more detailed information about the structure of the target, it is more flexible and reliable to simply ensure that the target is seen in multiple poses; this adds slightly more time for the data collection phase, but ultimately improves the calibration results with little effort.

Finally, different sensors have different sampling frequencies (even sensors of the same modality). For example, one camera may run at a higher resolution but lower frame rate, while another runs at lower resolution and higher frame rate, and a laser scanner or 3D laser may run at an even lower rate depending on the implementation (e.g., 0.1 Hz vs. 30-100 Hz). Dealing with varying time scales is not only a challenge during run time, but also at calibration time! However, at calibration time, we have more control over the situation. For example, we can tell the robot when to capture the calibration object (i.e., by sending a trigger message), and this could give it enough information to be able to record corresponding snapshots of data from each sensor. Note that the time taken to capture a frame (we call the set of all participating sensor readings captured after a trigger a "frame") cannot be any shorter than the period of the slowest sensor (let the slowest sensor period be T_{slow}). This is not the end of the story though, since the reason for the sensor's slow period is likely due to integrating sensed data over time. This means that unless the trigger is given at exactly the start of a sensing cycle, it's only safe to take the **second** reported data packet, since the first may have been incomplete and not even contain the calibration target. Therefore, the capture rate is no faster than $2T_{\text{slow}}^{-1}$. For sensors with sub-second capture periods, this is usually not a problem.

4.1.2 Summary

In this section, I have discussed why it is useful for robots to have as many sensors as possible and how multiple sensor modalities can enhance the ability to perceive the environment. I have also discussed the problem of fusing data from multiple sensors, and how knowing their relative pose is paramount to the fusion process. In addition,

I have explained why the problem is difficult yet of very practical importance.

In the rest of this chapter, I describe our flexible framework for accurately calibrating and fusing data from an (almost) arbitrary configuration of camera, laser, and point cloud sensors, where initialization estimates are generated automatically from the collected data. In addition, our system for collecting data requires only one operator, who needs minimal domain-specific expertise and need not tediously associate data points by hand. I describe a unique formulation of the calibration problem as a hypergraph containing both sensors and observations as separate vertices, incorporating geometric constraints between the different detection modalities. Finally, we developed a simulation system that produces all three types of data used for the calibration and can be used to produce benchmark data sets for future calibration research.

4.2 Solution

We call our solution the multi-sensor graph calibration (MSG-Cal) framework. MSG-Cal handles cameras, 2D laser range finders, and 3D sensors that output point clouds. It also handles any number of sensors, with the only constraints that: (1) every sensor field of view overlaps at least one other sensor's point of view enough to see the calibration object simultaneously, and (2) that 2D lasers must interact with at least one other modality (camera or 3D)[3]. These constraints are very reasonable, but do imply that MSG-Cal cannot handle the extrinsic calibration of systems with non-overlapping fields of view. However, MSG-Cal is a very modular system, and has the capability of handling new sensors and data collection schemes, simply by initializing the graph with the previous calibration and collecting calibration object observations from the new sensor in concert with the existing system.

[3]6-DoF 2D laser to 2D laser extrinsic pose calibration is difficult (at least when using a simple planar calibration target). Since the laser intersects the plane as a line, the single observation does not fully constrain the pose of the plane; there is a degree of freedom of rotation of the plane around the line. Since we assume we know nothing about the transform between the sensors (indeed, this is what we want to determine) and we cannot fix the plane pose relative to one of the sensors, we cannot use multiple observations of the plane to constrain the pose (i.e., we have fewer constraints/equations than unknowns).

MSG-Cal conceptually simplifies the problem by interpreting sensor readings as observations of one of two kinds of geometry: planes and lines. Cameras can detect planes through the use of fiducial markers that have a known size (e.g. AprilTags [155] or checkerboards), and 3D lasers and other point cloud sensing devices can detect planes directly by grouping points together that support a specific model of a plane. 2D lasers, on the other hand, can observe the lines that occur with the intersection of the target plane with the plane of the laser scan. If we can associate the observations of a single plane over two or more sensors, and we have multiple observations of this plane in different poses, then we can compute the relative poses of the pairs of sensors observing it. While we don't know the actual pose of the target object, this is not required; all we need to determine is the transform between a pair of sensors. Each observation provides a new constraint on the relationship between the sensors (usually; an observation may be removed as an outlier during the RANSAC-based pairwise calibration procedure).

In addition, MSG-Cal also provides methods and tools to make it easy to configure a system for calibration, collect data, and run the calibration process. The algorithm proceeds in three stages: data collection, target detection, and calibration. The first stage is obvious, but MSG-Cal provides two features that help make it more robust for multiple-sensor, multiple time-scale configurations: background subtraction and manual triggers. Background subtraction is used for the 2D and 3D lasers and any other point cloud producing sensor. This greatly simplifies the data association problem with very little cost: a small portion of the initial data collection is used to collect the background (meaning the calibration object should not be in view), and then the robot and background should stay relatively static during the remainder of the collection process. Some dynamic objects are tolerated, with both the assumption that they will not be as planar as the target object, and by using random sampling consensus (RANSAC) to minimize outliers in the pairwise calibration stage. Triggers are a particularly practical way to indicate to the system that it is time to collect data across sensors with varying time scales. The trigger indicates that the calibration object is ready to be recorded by all sensors, and the system captures the most recent

complete observation for all sensors into a single sensor frame.

The second phase is target detection. In this phase, each frame collected previously is processed to detect and compute the geometric properties of the target for each sensor (plane or line), determining the sets of valid pairs for each frame. Imaging sensors by default use the AprilTag detector for computing a plane estimate, 2D laser scanners subtract the background from the scan to yield candidate a candidate line, and the 3D lasers and other point cloud sensors run a standard RANSAC-based plane estimation process after filtering the background-subtracted cloud to reduce noise and lower the point count. The output of this phase is an organized set of pairwise detections consisting of plane-to-plane observations or plane-to-line observations.

The third and final calibration phase is itself split into two sub-phases. The first sub-phase uses each set of pairwise calibrations to estimate the sensor to sensor transform using RANSAC to filter outliers. This occurs for each pair of sensors that had a sufficient number of overlapping observations of the target object. The second sub-phase constructs a global hypergraph consisting of the estimated sensor poses, edges from sensors to observations, and constraints between cliques of sensors that observed the target object at the same time. We use the graph to construct and solve a non-linear optimization problem that yields optimized poses to minimize the error over all sensors.

4.3 Implementation

In this section, we discuss the implementation of the extrinsic calibration of multiple sensors with various modalities and acquisition timescales that are rigidly mounted to a robotic vehicle.

In this context, calibration means the estimation of relative sensor poses such that features detected by multiple sensors can be fused into a single coordinate frame. Due to sensor noise and systematic feature estimation uncertainty, it's unlikely that features will align without error, so the process must be constructed to mimimize the error across all sensors.

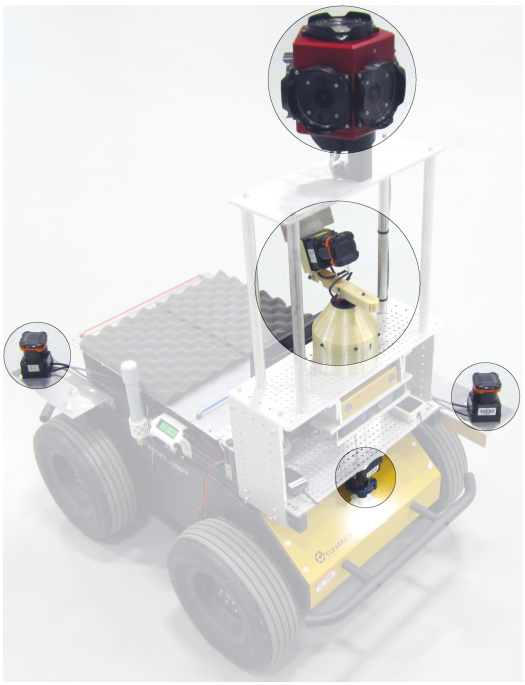


Figure 4.2 The robot sensors we calibrate. The image highlights the three 2D Hokuyo LRFs, the spinning Hokuyo 3D LRF, and the Flir Ladybug5 spherical camera.

In our case, we consider visible light cameras, 2D planar laser range finders (LRFs) such as the Hokuyo™UTM-30, and 3D laser range scanners (such as the Velodyne™HDL-32), or moving 2D LRFs such as the one shown in Fig. 4.2. We work with two geometric objects over the three sensor types: 3D lines derived from 2D LRFs and 3D planes derived from both the cameras and 3D LRFs. The calibration target itself is a large planar poster (as seen in Fig. 4.4a) with fiducial tags for the cameras. Section 4.3.1 describes the object detection methodology for each sensor type.

The process proceeds in four steps, described in the following sections.

4.3.1 Data collection

Data collection proceeds by launching a capture process with access to the robot's sensor streams. The user physically places the calibration object in various positions around the robot to capture sufficient views across all the sensors. Unfortunately, the sensors we are calibrating do not all support time synchronization. Therefore,

to effectively bypass the problem of temporal data association, we require the user to hold or place the board in a fixed position until all sensors have taken at least two measurements. The requirement for two measurements is for sensors with long integration times (several seconds), where one cannot be assured the board was not in motion during the capture of the first reading. This implies that the time the calibration object must remain still is at most twice the period of the slowest integrating sensor, which for most sensors will be less than one second. Thus, there is no need to specify the data rates, or provide synchronization between the sensor acquisition processes. Since the system receives messages at each datum from a sensor, it automatically provides an audible notification to the user when a frame is successfully captured.

We use the term *message* to refer to a single logical datum from a sensor, i.e. an image from a camera, a scan from a 2D LRF, or a point cloud from a 3D LRF. The term *frame* is used to refer to a collection of messages from every sensor. When the calibration object is in place for a new frame, the user sends a trigger to the system and data collection begins.

Each frame collected contains *all* the sensor data. While it may be possible that every sensor observes the calibration object, more likely only a subset of the sensors see it at one time. We must detect and extract the calibration object geometry for each sensor in order to correctly produce the edges in the sensor graph. At this stage, the system has no knowledge of the relative sensor positions, nor of the location of the calibration object in the environment, so given an image, point cloud or laser scan, the system must automatically detect the object for each sensor if it is in view.

4.3.2 Target Detection

Background subtraction

Since we rely on explicit correspondences between calibration object detections across sensors, we would like to have some assurance that the object we detect for each sensor in a frame is in fact the calibration object. We solve this problem using a background

subtraction approach, which practically eliminates the data association problem and avoids constraints related to sensor modality.

At the very beginning of data collection, we allow the point cloud and line sensors to build up a model of the background environment in order to *subtract* it from the live dynamic data generated by the calibration target. This technique is typically used in imaging systems for surveillance, object tracking, and background replacement [106]. In our algorithm, we use an analogous technique for 3D point clouds provided by PCL [179] that takes advantage of a modified octree. This octree is a spatial data structure that hierarchically subdivides volumes into octants and also provides efficient change detection. We accumulate points from the 3D sensors over a short period of time, and store the observations in the octree. When data collection begins we create a new octree for each message from the sensors, and then subtract the accumulated background, leaving only the calibration operator and target (and possibly a few other noisy points) in the scene. Due to our usage of AprilTag fiducials on the calibration object, we do not need any explicit technique for background subtraction for cameras, since the tag detection automatically takes care of that.

Line and plane extraction from point clouds

Given a sparse point cloud from the background subtraction process, we use the following process to find the calibration object plane. First, sparse points are removed using a statistical filter, normals are computed for each point using the local neighborhood and then clusters are discovered by growing regions seeded by a point with local minimum curvature. For each cluster, the largest plane is extracted using RANSAC. The plane that best matches the known dimensions of the calibration object is selected as the valid detection for this sensor. In practice, there are few clusters due to the background extraction (usually related to the user's body or other stray points in the environment that may not have been captured by the background accumulation), and all but one of these clusters is too small to be the object.

Line extraction from the 2D laser range finders proceeds in much the same way as the plane extraction, however, instead of using a plane model, we use a line model with

RANSAC. For each detected line, found using loose thresholds to enable high recall, we ensure it is contiguous and within the required size bounds (given an approximate size of the calibration object). Finally, we filter the egregious outliers, then refit a line to the remaining points. The endpoints we use to define the detected line segment are determined by projecting the actual scan endpoints to the closest point on the model line. The resulting line segment is reported as the detection for this sensor.

Plane extraction using AprilTags

As described in section 4.3, the calibration target is primarily a planar object used to induce lines and planes from 2D and 3D LRFs, but we need to also be able to detect the plane from a camera. In the general case, cameras are projective devices, and can only determine an object pose up to scale. Therefore, the AprilTag algorithm utilizes the *known* size of the tag, planar homography, and the camera parameters to determine the 3D pose of each tag [155], [194]. We utilize three tags for redundancy, and use as many tags as are visible to determine the plane parameters. If more than one tag is visible, we currently compute the average normal and distance to origin.

4.3.3 Pairwise calibration

The pairwise calibration phase estimates the $SE(3)$ transforms between each pair of sensors with co-occuring detections, using RANSAC to filter outliers. Since we convert all detections into the geometric primitives of either planes or lines, then in order to estimate the transform between each pair, we must solve one of the following objective functions.

Plane to plane

We use the Hessian normal form of a plane $P_i = \{\hat{\mathbf{n}}_i, d_i\}$ where $\hat{\mathbf{n}}_i^T \mathbf{x} = -d_i$. Therefore, if $\mathbf{x}$ is a point on plane P_i, then $\hat{\mathbf{n}}_i^T \mathbf{x} + d_i = 0$. As in [202], we note that the rotation and translation are separable problems. The normals are related by the rotation ${}^iR_j$ alone: $\hat{\mathbf{n}}_i = {}^iR_j \hat{\mathbf{n}}_j$. Using the plane equation, we can define the relations:

$$\hat{\mathbf{n}}_i^T({}^iR_j\mathbf{x}_j + {}^i\mathbf{t}_j) = -d_i$$

$$\hat{\mathbf{n}}_i^T\,{}^iR_j(-d_j\hat{\mathbf{n}}_j) + \hat{\mathbf{n}}_i^T\,{}^i\mathbf{t}_j = -d_i$$

$$-d_j\hat{\mathbf{n}}_i^T({}^iR_j\hat{\mathbf{n}}_j) + \hat{\mathbf{n}}_i^T\,{}^i\mathbf{t}_j = -d_i \tag{4.1}$$

$$\hat{\mathbf{n}}_i^T\,{}^i\mathbf{t}_j + d_i - d_j = 0, \tag{4.2}$$

where, in Equation 4.1 we utilize the fact the $\hat{\mathbf{n}}_i^T\,{}^iR_j\hat{\mathbf{n}}_j \approx 1$, since the corresponding normals are unit vectors and ${}^iR_j$ brings $\hat{\mathbf{n}}_j$ into the frame of $\hat{\mathbf{n}}_i$.

We use these constraints in the following objective function:

$$\min_{({}^iR_j,{}^i\mathbf{t}_j)\in SE(3)} \sum_{(P_i,P_j)\in\mathcal{C}} \left\|\hat{\mathbf{n}}_i - {}^iR_j\hat{\mathbf{n}}_j\right\| + \sum_{(P_i,P_j)\in\mathcal{C}} \left[\hat{\mathbf{n}}_i^T\,{}^i\mathbf{t}_j + d_i - d_j\right]^2, \tag{4.3}$$

where $\mathcal{C}$ is the set of plane correspondences for a sensor pair. In our implementation, we use the Kabsch algorithm [104] for estimating the rotation between the normal sets. To determine the translation, we compute the least squares result of Equation 4.2. Initially, we treat all the normals with equal weight. However in the future we plan to compute the uncertainty of the detected planes and use a variant of Wahba's algorithm to compute the optimal rotation as per the method discussed in [169].

Plane to line

In contrast to our previous work [202], we utilize an objective function that minimizes the distance of the segment endpoints to the corresponding plane:

$$\min_{(R,t)\in SE(3)} \sum_{(P_i,\ell_i)\in\mathcal{C}} \sum_{\mathbf{x}_i^j\in\ell_i} \left[\hat{\mathbf{n}}_i^T\mathbf{x}_i^j + d_i\right]^2 \tag{4.4}$$

While we provide no initial estimate, we have discovered that the line to plane RANSAC algorithm performs best with a sample size of five in order to sufficiently

constrain the degrees of freedom.

4.3.4 Global calibration

We now describe the graph optimization that achieves the global calibration. In the previous phase, we compute a pairwise relative pose for every pair of sensors that meet or exceed the minimum number of observations. In this phase, we construct a hypergraph composed of several node and edge types that exploit the pairwise relative transforms as an initialization for the global sensor pose graph. The distinction is subtle but important: in the robot frame, we must pick one sensor as the root of the transform hierarchy (most transform libraries, e.g. `tf` in ROS, require a transform tree), and then connect the other sensors to this root, as per the pairwise connections in the previous phase, effectively forming a spanning tree over the graph of sensors. In addition, the pairwise calibration phase uses no additional information (e.g., other co-occuring detections) in order to minimize the observation error over more than one path in the graph. The goal of the global graph approach is to incorporate all the information into a single optimization structure, courtesy of g2o [119]. In our formulation, the sensor poses are the unknowns we wish to estimate simultaneously in a global frame. We let the user pick one sensor that will act as the root of the graph (e.g. a sensor with the largest FOV) and therefore become the origin of the global frame. In order to estimate the initial poses relative to the root, we construct a minimum spanning tree $T_{\mathbb{S}}$ based on the edge weight between the sensors, defined as the sum of the squared errors computed during the pairwise calibration.

We define our hypergraph G as a tuple of nodes V and hyperedges E. We include three node types in $V = \mathbb{S} \cup \mathbb{L} \cup \mathbb{P}$: sensor ($\mathbb{S}$), line-observation ($\mathbb{L}$), and plane-observation ($\mathbb{P}$). The sensor node set $\mathbb{S} \subset SE(3)$ contains elements $\mathbf{x}^{\mathbb{S}} \in \mathbb{S}$, the unknown sensor poses, initialized as per the sensor-sensor transform computed from the spanning tree $T_{\mathbb{S}}$. The line-observations and plane-observations correspond to every inlier detection found during the pairwise calibration phase. Line-observations $\mathbb{L} \subset \mathbb{R}^3 \times \mathbb{R}^3$ are elements $\mathbf{x}^{\mathbb{L}} \in \mathbb{L}$, the endpoints of the detected lines. Plane-observations $\mathbb{P} \subset \mathbb{R}^4$ are elements $\mathbf{x}^{\mathbb{P}} \in \mathbb{P}$, i.e. the normal and distance to the origin.

Note that $\mathbb{L}$ and $\mathbb{P}$ nodes exist relative to their observing sensors, while the sensors are the only entities in the global frame (where the root of $T_{\mathbb{S}}$ is taken as the origin).

The objective of our graph optimization is to find the sensor pose set $\mathbb{S}$ that minimizes the error over all edges in the graph. One can think of the nodes $\mathbf{x}_i^\alpha \in V,\ \alpha \in \{\mathbb{S}, \mathbb{P}, \mathbb{L}\}$ as providing the data values where i is the node identity and α is the node type. The hyperedges $e_\gamma \in E$ provide relations on the data, where $\gamma \subset \{i : \mathbf{x}_i^\alpha \in V\}$. The edges are the observations from the sensors and we construct functions F_{e_γ} that represent the noisy constraints in the system. Since the graph contains three node types there are necessarily six corresponding binary edge types, however, we currently only make use of five since we have not yet implemented a line-to-line pairwise calibration procedure.

The functions F_{e_γ} compute the errors we wish to minimize, representing the degree to which the parameters $\mathbf{x}_i^\alpha, \ldots, \mathbf{x}_j^\beta$ satisfy the initial constraint μ_γ (computed as the edge type-specific sensor observation). $\Omega_{\alpha,\beta}$ represents the information matrix of the constraint.

$$E(G) = \min_{\mathbb{S}} \sum_{e_\gamma \in G} F_{e_\gamma}(\mathbf{x}_i^\alpha, \ldots, \mathbf{x}_j^\beta, \mu_\gamma)^\top \Omega_{\alpha,\beta} F_{e_\gamma}(\mathbf{x}_i^\alpha, \ldots, \mathbf{x}_j^\beta, \mu_\gamma) \tag{4.5}$$

The graph optimization is implemented as a sparse non-linear least-squares minimization over the error values produced by the graph edges as shown in Eq. 4.5. During optimization, the algorithm computes the Jacobians of the error functions, and takes small linear steps in the tangent space of the sensor node manifold, in order to distribute the error over the nodes as defined by their uncertainty. Each node type provides an appropriate step operator in the tangent space, and uses the exponential map as necessary to compute the corresponding point on the manifold.

In the following sub-sections, we define each error function F_{e_γ} given the edge type defined by the set of observation types in each node.

Sensor-sensor edge

The sensor-sensor edge represents the relative transform between the sensor nodes it relates. Our error is defined as the difference of the relative transform from the mean. We define the mean to be the initial relative transform we compute from the pairwise calibration phase for this pair of sensors. If $\mu_{ij} \in SE(3)$ is the initial relative transform between sensor pair $\mathbf{x}_i^{\mathbb{S}}$, $\mathbf{x}_j^{\mathbb{S}}$, then the edge error in the tangent space is defined as:

$$\log_{SE(3)}(\mu_{ij}^{-1}((\mathbf{x}_i^{\mathbb{S}})^{-1}\mathbf{x}_j^{\mathbb{S}})). \tag{4.6}$$

Sensor-line edge

The sensor-line edge represents the measured line endpoints relative to the sensor's coordinate frame. This edge constrains the adjustment of the line in the sensor frame with respect to the original observation. The edge uncertainty is related to the noise inherent in the laser scan, i.e. we assume the points (r, θ) detected by the 2D LRF are perturbed by two independent normally distributed variables a, b as in:

$$p_i = (r + a, \theta + b) \tag{4.7}$$

$$a \sim \mathcal{N}(0, \sigma_r) \tag{4.8}$$

$$b \sim \mathcal{N}(0, \sigma_\theta). \tag{4.9}$$

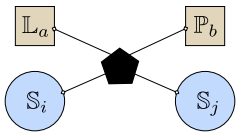
The error is defined in a six dimensional space ($\mathbb{R}^3 \times \mathbb{R}^3$), with extremely low uncertainty in the z coordinate for each point. If μ_{ij} is the initially observed endpoints of the line j from sensor i, then the error is $\mathbf{x}_j^{\mathbb{L}} - \mu_{ij}$.

Sensor-plane edge

The sensor-plane edge is defined similarly to the sensor-line edge, but is parameterized by the four dimensional space ($\mathbb{R}^3 \times \mathbb{R}$) representing the plane parameters. Given μ_{ij} as the initially observed plane parameters, the error is $\mathbf{x}_j^{\mathbb{P}} - \mu_{ij}$.

Observation-observation edges

The following two hyperedges correspond to the unique constraints formed by the interaction between the geometric primitives. Each hyperedge is structured in the following manner:



where the squares are the observation nodes (line,plane) and the circles are the sensor nodes. The black pentagon represents the hyperedge. In every case, the observations are in the local frame of the corresponding sensor, which requires that we transform one of the objects into the frame of the other sensor. This is achievable since the sensor nodes represent their pose estimate in the global frame. We can compute the relative transform between the sensors and use this to bring the second observation into the frame of the first.

Line-plane edge

The line-plane edge connects a line and plane as detected by the respective sensor. In this case, we use the plane definition to represent the error in $\mathbb{R}^2$ as the offset of each of the line endpoints to the plane.

Plane-plane edge

Finally, the plane-plane edge connects the observations of the same plane from two different sensors. Given the transform of P_2 into the frame of P_1, we define the error in $\mathbb{R}^4$ as $(a_1, b_1, c_1, d_1) - (a_2, b_2, c_2, d_2)$, where a, b, c, d are the plane parameters.

4.4 Evaluation

To evaluate the algorithm, we develop a simulation framework to provide ground truth for evaluating the correctness of the algorithm, and we collect real data for qualitative evaluation using color camera fusion with laser-generated 3D point clouds.

4.4.1 Simulation

In order to provide a benchmark quantitative evaluation of the proposed calibration procedure, we developed a simulation that generates the same raw sensor messages as the robot yet allows us to specify the ground truth poses for the sensors. The simulation includes a geometric representation of the calibration object derived from the generating PostScript program and realistic implementations of the target sensors: a pinhole camera that produces images of the object, a 2D laser range finder to generate point clouds from the (r, θ) scan, and a spinning 2D laser range finder that produces 3D point clouds. We briefly describe the data set generation process, then follow with the implementation details of the sensors.

Data set generation

Similar to the collection procedure described in Section 4.3.1, we generate randomly distributed poses for the calibration object, assuming the sensor system is situated at the origin. The procedure allows one to specify the bearing θ, radius r, maximum rotation around an axis ϕ and the number of random calibration object rotation samples to generate at each pose (θ, r),

$$\begin{aligned}\mathbb{R}_{(\theta,r)} =&\{(\psi_r, \psi_p, \psi_y)^i \mid \psi_r \sim U(-\phi_r, \phi_r),\\ &\psi_p \sim U(-\phi_p, \phi_p), \psi_y \sim U(-\phi_y, \phi_y)\}.\end{aligned}$$

For each sample, we "expose" the transformed calibration object to each simulated sensor and record the observations in a bag file.

2D laser range finder

We implement the 2D LRF by intersecting rays cast from the origin of the sensor with the plane of the calibration object. Assuming the origin of the sensor is p_o, we construct a set of rays $\{\rho_i\}_1^N$ (normalized vectors) emanating from p_o with a configured angular resolution, start, and end angle. For each scan, we perturb the angles used to generate the rays, and once a valid intersection is found, we perturb the true radius

according to Equation 4.9. We use the following ray intersection equation:

$$\hat{\mathbf{n}} \cdot (\rho_i t + \mathbf{p}_o) + d = 0 \tag{4.10}$$

$$\hat{\mathbf{n}} \cdot \rho_i t = -\hat{\mathbf{n}} \cdot \mathbf{p}_0 - d \tag{4.11}$$

$$t = \frac{-\hat{\mathbf{n}} \cdot \mathbf{p}_o - d}{\hat{\mathbf{n}} \cdot \rho_i}, \tag{4.12}$$

and discard intersections where $t < 0$, since that means the intersection is behind the laser. With a value for t we can compute the point on the plane, and then determine whether it is within the calibration object bounds. This is accomplished by testing for negative intersections with all four half-planes defined by the boundary segment normals. These normals lie orthogonal to the plane normal and boundary segment, and emanate away from the centroid. Figure 4.4b shows an example laser scan in 3D.

3D laser range finder

The 3D LRF is implemented by "spinning" the 2D LRF around a vertical axis and therefore generating multiple 2D laser scans from different poses. We mimic the construction of the sensor shown in Fig. 4.2, with the following parameters: tilt, offset, scans per degree, and scan angle. The first two parameters control the structure of the sensor, while last two parameters determine the density of the scans and the field of view (see Fig. 4.3 for an illustration of the geometry). If one sets the offset and tilt both to zero, then the effect would be to simply rotate a 2D LRF with the forward axis parallel to the z axis. Since the pose of the scan plane changes as the LRF rotates around the axis, we must transform each scan into the frame of the origin. Doing so generates the blue cross-hatched plane shown in Fig. 4.4b.

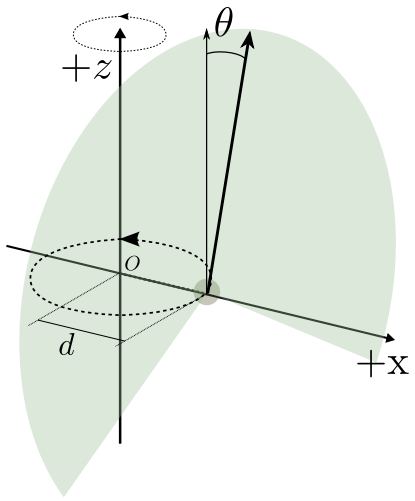


Figure 4.3 The structure of the 3D laser range finder. θ indicates the tilt (away from positive z), while d controls the offset from the axis of rotation, positive z. The shaded region indicates the vertical scan pattern from the 2D LRF.

Pinhole camera

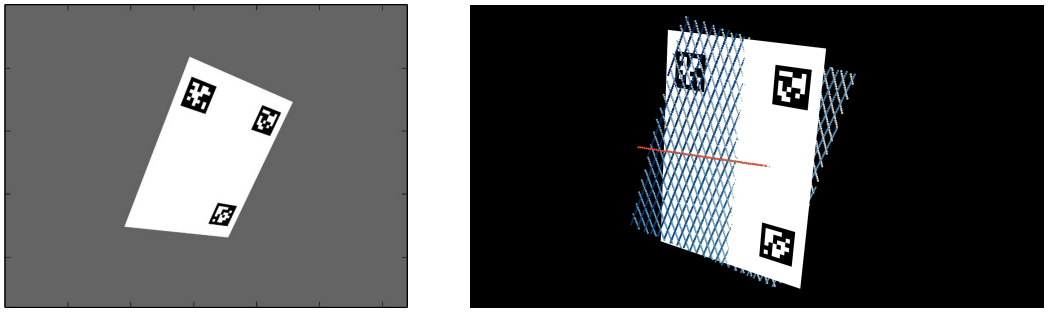
The camera is modeled by the width and height of the generated image and the camera matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.13}$$

We make the assumption that the camera is intrinsically calibrated, and do not generate any distortion. The image generation process is as follows: (1) a high resolution image of the calibration object is generated (i.e. 3 pixels per mm) directly from the metric specification of the poster size and embedded AprilTags. (2) Boundary coordinates are extracted from the transformed points of the calibration object, and are projected onto the image plane using the projection $\pi : \mathbb{R}^3 \to \mathbb{R}^2$:

$$\pi(\mathbf{x}) = \left(\frac{f_x x_1}{x_3} - c_x, \frac{f_y x_2}{x_3} - c_y \right). \tag{4.14}$$

Given the real image coordinates, we compute the homography matrix H that transforms the high-resolution model image to the planar deformation in the projected



(a) (b)

Figure 4.4 (a) An image generated from a rotated calibration object at 2 m. (b) An example laser scan and point cloud rendered in 3D before calibration, along with the ground truth object

image. An example image is shown in Fig. 4.4a.

4.4.2 Experimental results

We evaluate the proposed system on two sets of sensors: one in which we know the ground truth, and can provide quantitative error metrics with respect to computed versus actual position, and a set of sensors mounted on a mobile robot, in which we do not have ground truth. The calibration system is implemented as a set of C++ and Python packages based on the ROS ecosystem. The simulated and real datasets contain 120 and 133 different calibration object poses corresponding to one frame per pose, respectively.

The robot is a ClearPath Husky™with five primary sensors as shown in Fig. 4.2. All three modalities are represented, including several 2D LRFs, one slowly rotating 3D LRF, and five cameras we use as one camera from the Ladybug5 spherical camera.

Error computation

We report our results using the following two metrics: error over the graph and deviation from ground truth (for the simulation study). The graph error is currently defined as the sum of the squared error over all edges other than the sensor-sensor edges (i.e. every sensor-observation and observation-observation edge). Thus, this error is related to *all* the data embedded in the graph. Every error is represented

in terms of meters, so the error unit is m^2. For the simulation, we directly compare the sensor pose model with the resulting poses from the optimization procedures. We report these errors as the mean translational and angular offset (measured as the smallest angle between the principal axes of two sensors, i.e. the angle subtending the geodesic on the unit sphere S^2) from ground truth over all sensors.

We also investigated several variations of the algorithm to determine performance under different conditions. In particular, we were interested in the value of the pairwise calibration phase. First, we compared the global result when the sensors were initialized from the pairwise spanning tree transform versus using the identity transform, and discovered this yielded no significant difference. However, in both cases the inlier set from the pairwise calibration was used. We wondered whether the results would be the same if we used all the data in the graph (no filter) vs. using the pairwise RANSAC inlier set (SAC filtered). The following two sections summarize our results for the simulation and real-world robot sensors.

Simulation

We use the simulation to show the algorithm works with respect to ground truth, given realistic simulated sensor data. Figures 4.5a and 4.5b show the benchmark error of the simulation configuration against the known ground truth sensor poses. One thing to note is that the variance of the graph solution is always lower than just the pairwise pose results over multiple runs. In addition, it is interesting to note that the mean of the pairwise angular error is lower than the graph (by 0.03 degrees), but this is not entirely unexpected given the error over the whole system: the pairwise solution represents the best relative pose given the data, but the global optimization has to correct for these errors across the entire graph. In Fig. 4.5c, we see that the global graph error mean and variance are reduced as compared to the *best* pairwise solution with no global optimization. Finally, Fig. 4.5e illustrates the performance over multiple runs while comparing the global optimization performance with and without filtering the outliers using the pairwise calibration phase. Note that while the filtered error shows small variance, the error is significantly lower when the

outliers are filtered.

Robot Sensors

To test the algorithm using real-world sensors, we collected data from the robot and sensors shown in Fig. 4.2. Since we do not have ground truth, we report the results of the graph error given the pose estimates from the pairwise calibration as compared to the error after the global graph optimization (Fig. 4.5d). We also show the difference between filtering the outliers and using all the data with no filtering (Fig. 4.5f). Note the agreement between the relative values as compared to the simulation results; graph optimization improves the error over the pairwise initialization, given the structure of the edge constraints. In addition, filtering the outliers significantly improves the error result.

However, we feel the qualitative results speak more clearly to the capability of the algorithm, and we show two screen captures from the 3D visualization of the fused data. Figure 4.6a shows a large indoor area captured with the system with an inset of the corresponding image from the Ladybug's cameras. Figure 4.6b shows another scene from a highly cluttered machine shop in the same building, with small details that reinforce the practical performance of the system.

4.5 Summary

We have presented an end-to-end system for calibrating a set of minimally constrained multi-modal sensors rigidly mounted to a robot. Although many of the individual concepts have been presented in previous works, as far as we are aware this is the first algorithm to bring them together into a cohesive graph optimization framework paired with a simplified data association process.

We have many plans to improve the capability of the MSG-Cal system as presented. In particular, some mobile robots may have sensors on dynamic joints (e.g., mounted on pan/tilt units) or wish to ensure a manipulator end effector is calibrated with one or more sensors. Therefore, we are currently in the process of adding dynamic

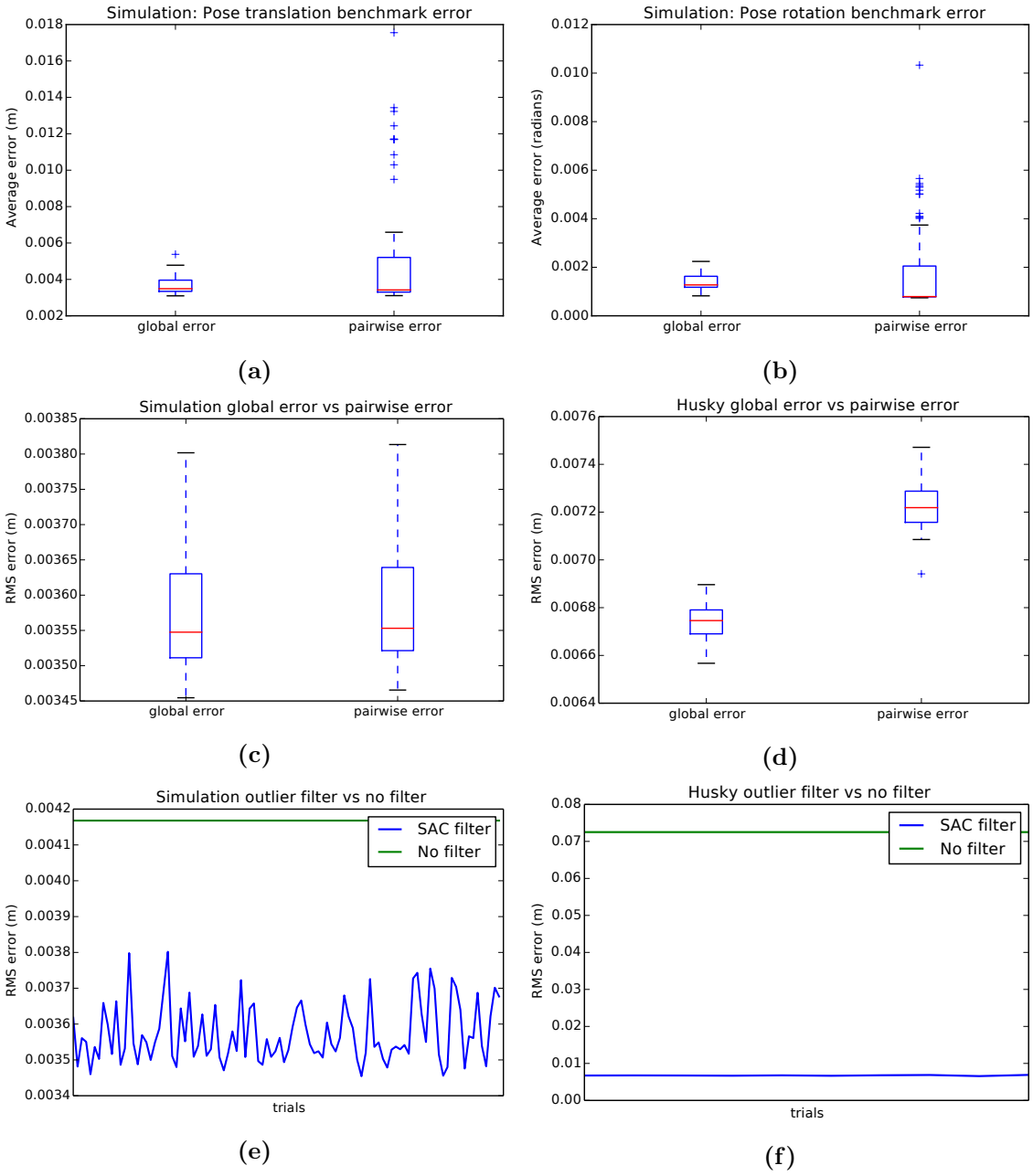
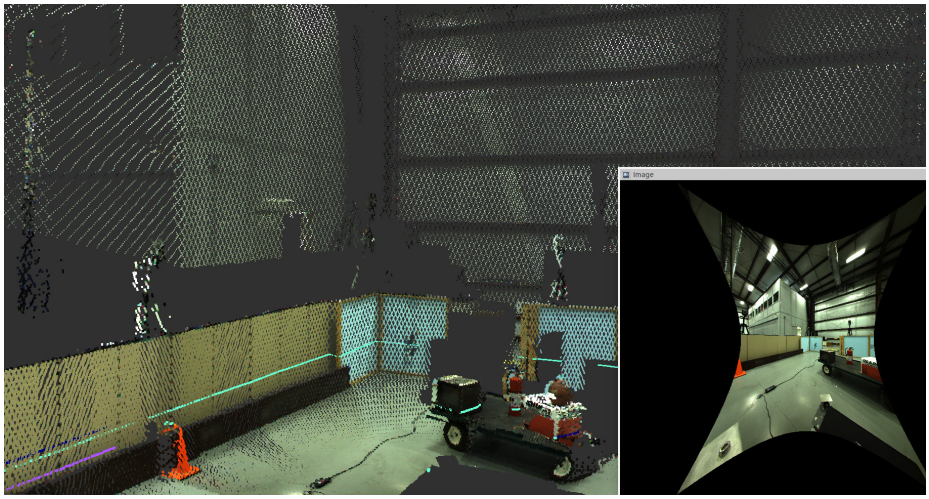
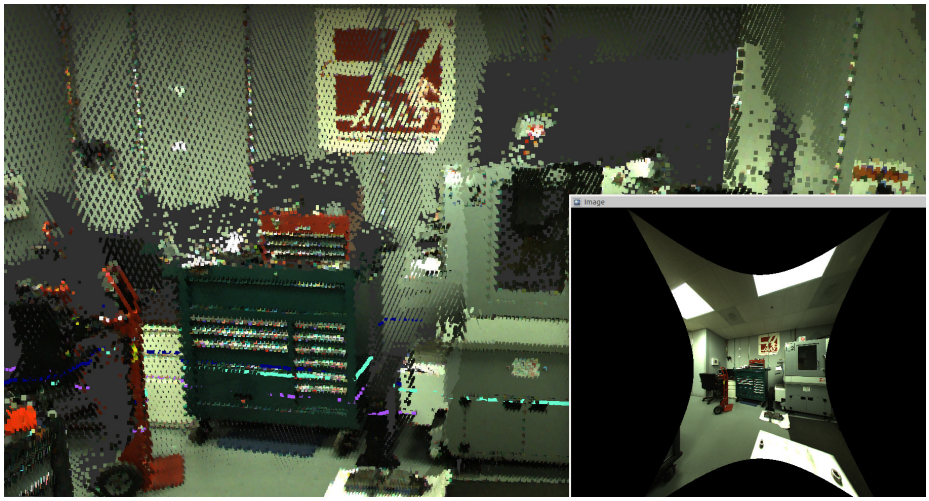


Figure 4.5 (a) Simulation mean translation benchmark error (poses computed against ground truth). (b) Simulation mean rotational benchmark error (poses computed against ground truth). (c) Simulation graph error comparing the best pairwise error result vs. global error. (d) Husky graph error comparing the best pairwise error result vs. global error. (e) Simulation global graph error comparing results with and without filtering the data using the sampling consensus. (f) Husky global graph error comparing results with and without filtering the data using the sampling consensus.



(a) Indoor, open area scan.



(b) Indoor, cluttered area scan.

Figure 4.6 Fused sensor output using the global calibration results for the Husky sensor data collection. The Ladybug images are back-projected to the point cloud, given the relative transform between the two sensors. Also note the blue, purple, and cyan points that show the transformed 2D laser scan output. In particular, (b) shows how well the laser scans align to the camera and 3D LRF, e.g. by their intersection on the handcart on the left side of the image.

joints to the graph and modifying the data capture process accordingly. Additionally the current system is based completely on visual sensors; while certain processes may have to change to support proprioceptive sensors such as IMUs, adding this functionality to the system would be worth some of the compromises for the robots that would benefit from it.

While we've already begun to evaluate the algorithm on a wider array of data sets, we also plan to explore improvements to the camera plane estimation algorithm and provide specific improvements to camera-camera pair calibrations.

Finally, my colleagues have begun the necessary work to make the code available as an Open Source distribution for ROS so others can benefit and improve upon the work.

Chapter 5

Ego-motion Estimation Using Vision

5.1 Introduction

A robot must know where it is and how it is moving for a variety of tasks, including localization with respect to a known map, building a new map while localizing, and determining its relationship to an object for manipulation e.g., opening a door, picking up a tool, and shoving an object out of the way. The task of estimating the motion of the robot's body is called ego-motion estimation.

There are a variety of ways for a robot to estimate pose. First, a wheeled robot may use knowledge of how the wheels turn to estimate how it is moving. This is called dead reckoning, or odometry (odos - way, metron - measurement). This is a very useful way to estimate motion, especially if the wheels have little slip and the surfaces the robot drives on are not very complex. However, some robots may slip on different surfaces, bump around, or use techniques for turning like skid steer that cause motion estimation based on measuring wheel motion to fail or, at the minimum, be inaccurate due to slippage and other un-sensed interactions with the ground (by accumulating enough non-recoverable error the estimate pose no longer represents the actual pose).

It would be useful to have a way of measuring motion based on how the robot pose

changes with respect to the environment, assuming large parts of the environment do not themselves change while trying to take the measurement. Using vision is one way to do this. Cameras are a relatively low cost sensor and they provide a lot of information about the environment that the robot may use to estimate motion. The basic operation of visual odometry (the approach of computing odometry using visual sensors) is simple to explain but is considerably more complicated to implement.

Thinking generally, how could we reliably determine ego-motion? Imagine we place a human or robot in the "construct," the infinitely white artificial loading area for the characters in the movie The Matrix. How would they recognize their motion? Since there are no features or structures in any way (maybe a floor, maybe not), there *is no way* to measure their motion with respect to the environment, because it doesn't really exist. While humans and robots can detect acceleration using their inertial measurement units (i.e. accelerometers and gyroscopes for a robot and inner ear for a human), without some kind of observable features in the environment, something to "measure" distances by, there would be no motion.

Thus, computing the motion of a camera simply from the data it provides is not a new concept. It has been called many things, often depending on the approach taken, but the general term of visual odometry is apt here since we wish to measure the motion of the robot system (which includes the camera) using the data produced by the camera. In this dissertation, we use an RGB-D camera, because it has an additional sensor and emitter that allow it to sense both color (red, green, blue: RGB) and depth (D). The way it computes depth is through a process called stereo disparity which computes the difference in location of an observed feature seen from two different, but known, viewpoints. See section 3.3.2 for a slightly more in-depth discussion.

When the color and IR camera are extrinsically calibrated with respect to each other (which we include in the intrinsic calibration of the RGB-D sensor *system*), it is possible to map a color to every depth pixel, and back-project the point into the world to generate a point cloud for every frame seen by the camera. Sensors such as the Microsoft Kinect and Asus Xtion Pro Live provide this capability over

a USB connection, and can be operated using open source software from OpenNI (conveniently integrated into ROS).

Point cloud generation is an ability most cameras do not possess, but in this case it affords us with a straight-forward frame to frame alignment methodology, shown in high-level pseudocode in Algorithm 1 [44].

Algorithm 1 Feature-based rigid motion estimation

1: compute features in query and model frame
2: compute correspondences between the features
3: P_q = point cloud of corresponding feature points in the query frame
4: P_m = point cloud of corresponding feature points in the model frame
5: $c_q = \mathbf{centroid}(P_q)$
6: $c_m = \mathbf{centroid}(P_m)$
7: $C = \sum_{q \in P_q, m \in P_m} (q - c_q)(m - c_m)^T$
8: $(U, s, V^T) = \mathbf{SVD}(C)$
9: $S = [1, 1, \det(VU^T)]^T$
10: $R = V\mathbf{diag}(S)U^T$
11: $t = c_q - Rc_m$

This algorithm utilizes image feature correspondences between frames to compute a rigid motion estimate that transforms the query cloud to the model frame. One aspect to note about this formulation is the requirement for image features: distinctive locations in the image that are robust to small changes in viewpoint and therefore can be reliably detected from a different viewpoint. The best features are expensive to compute yet still rely on having "interesting" image content. This can be problematic when the image has repetitive textures, mildly textured regions, or high framerate processing is required (e.g. for robot motion estimation).

The purpose of the descriptor is to provide a hopefully unique description of that point within the image; usually this requires looking at a small area surrounding that particular point in the image in order to derive enough context to provide a usable and comparable description. The computation of correspondences, line 2 in Algorithm 1, will not work well without eliminating outliers by embedding it in a robust estimation framework like RANSAC. This implies evaluating many iterations of the algorithm with random corresponding samples and choosing the correspondence set that maximizes the number of inliers.

While this algorithm utilizes both visual and structural features in turn, if enough features are not found to generate sufficient correspondence between successive frames, then the algorithm fails and loses tracking. For a purely visual system, this is unavoidable in some cases. For example, attempting to track motion against a blank wall with no discernable features (often found indoors) will cause almost every algorithm to fail; only systems that can rely on proprioceptive information like an IMU can continue estimation during these failures. However, if we consider an algorithm that uses both structural and visual information, we can handle many, if not all, situations that would cause a feature-based algorithm trouble.

Following Whelan et al. [213], our approach merges two techniques that combine structural alignment with photometric alignment in order to estimate the small camera motion between frames. We combine an iterative closest point algorithm using the point cloud data with a dense visual odometry algorithm that uses both the point cloud and image intensities.

Each algorithm is independently executed on the frame pair to generate intermediate constraints which are then combined to form a single set of weighted normal equations. We exploit the massively parallel SIMT (single instruction, multiple threads) capability of the GPU to compute both the ICP and DVO steps. The output of this portion of the algorithm is an estimate of the camera motion between frame t and $t + 1$.

5.2 RGB-D Sensors for Visual Odometry

RGB-D sensors are useful as a visual odometry modality since structured light or laser ranging can generate features where there are none for computing depth. Also, in contrast to most stereo algorithms (although it depends on the environment and the amount of time you are willing to wait) RGB-D sensors typically generate dense depth. Commodity RGB-D sensors provide color RGB images as well as depth, and have internal calibration parameters that allow the color and depth to be aligned. We discuss this in further detail in Chapter 3 on sensor modalities.

We use structured-light RGB-D sensors because we wanted to take advantage of the appearance information present in the color information, while also exploiting the density and information inherent in the depth image. By combining an ICP algorithm with dense visual odometry [107], [192] we can exploit the best complementary features from both.

5.2.1 Feature-based Approach

In this section, we discuss our initial approach to this problem, based on the standard frame to frame feature matching algorithm (see Alg. 1). We processed the RGB images in several different ways to find features and compute descriptors. Using an RGB-D camera, this approach provides some arguable improvements to monocular visual odometry methods as the sensor itself provides metric measurements that (almost) immediately determine the actual position of the feature points in the world relative to the camera pose. I qualify the statement with "almost," since RGB-D cameras still suffer from image noise and inaccuracies in stereo matching that don't always give the correct depth for a particular feature.

After finding feature points in the current image, the next step is to find correspondences between these points and the points in the previous frame. To do this, we compute a descriptor for each feature detected in the image. The aim of a descriptor is to provide a hopefully unique numerical *description* of the detected location within the image. Descriptors are often specified as some n-dimensional vector of reals, so $\mathcal{D}_i \in \mathbb{R}^n$. Usually this requires looking at the area surrounding the point in the image in order to derive enough context to generate such a description. The goal of a descriptor algorithm is to be robust to various changes in lighting, scale, and rotation so that if $\mathcal{D}_i$ and $\mathcal{D}_j$ are generated from the same world point, then $d(\mathcal{D}_i, \mathcal{D}_j) < d(\mathcal{D}_i, \mathcal{D}_k)$ for some distance metric $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, and where $\mathcal{D}_k$ is a descriptor for an unrelated feature in the image.

This is a whole field in itself; detailing the creation of efficient feature detectors and effective feature descriptors for the primary goal of obtaining correspondences between two or more images can fill a book. Feature detection and description are

also useful for tasks outside of visual odometry whenever there is an interest in finding relationships and correspondences between a set of images, e.g., stitching photos into panoramas or localizing image location with respect to other cameras. In the case of visual odometry, the descriptors are used to find the putative feature matches between two images, by selecting matches that have the smallest descriptor distance between them.

After generating an initial set of matches, it is quite likely there are some matches that are not valid. In this case, we need to use one or more schemes to try and find the best set of matches. One way of doing this is by randomly sampling the correspondences and attempting to fit a model to the set of correspondences. For ego-motion estimation, this model is the transformation $T_i^j = \left(\mathbf{R}_i^j, \mathbf{t}_i^j\right)$ that represents the motion of the camera between frame i and frame j. This model is then used to transform the feature locations from frame j to the coordinate frame for frame i, and then use either a Euclidean distance or reprojection error metric to determine match quality. Selecting a threshold for this metric allows us to exclude correspondences that don't meet the criteria while including correspondences that do meet the criteria. Note that this step effectively disregards the descriptors computed initially, and provides a geometric constraint on the feature match.

This process is called RANSAC [61]. In RGB-D visual odometry we can use a 3 point RANSAC, i.e., we can sample three point correspondences, since each one of our feature points has depth, allowing us to back-project the image point to an $[X, Y, Z]^\mathsf{T}$ Cartesian position within the world. Thus, we only need three (non-colinear) points to determine the full 6 degree of freedom (DoF) transformation between the corresponding points. Using the transformation estimated from the sample, we transform the entire feature point set and compare nearby points. This requires some care since there is still error in the point positions as well as noise in the image contributing to inaccuracies in the backprojected feature locations. Even small errors in feature location are exacerbated during the back projection process if the point has a large depth value, therefore some combination of Euclidean distance and reprojection error can be used to mitigate this effect.

We perform the sampling, estimation, and error computation for some number of iterations, until we are confident we have found the best (or "good enough") set of correspondences. If you know (or estimate) the probability of finding a sample that is part of a good model is p_g, and the probability the algorithm will exit without finding a good fit *if one exists* p_{fail} given the size of the sample N, then you can compute an estimate for the number of iterations L: $L = \frac{\log(p_{\text{fail}})}{\log(1-(p_g)^N)}$. Data that match the sampled model are called inliers, while the rejected correspondences are called outliers.

When we find a model that has a high enough number of inliers, we can use the estimated transformation directly, or refine the transformation estimate using the full inlier set of correspondences. The camera motion is the inverse of the estimated point transform. Setting the camera to the newly computed pose and capturing another image, we can iteratively construct a path that represents the motion of the camera through the scene. This is the simplest way of performing feature-based visual odometry in RGB-D.

We would like to note several things about this approach. First, the success of the approach is wholly dependent on the quality of the features and the ability to match the features between successive frames. Therefore, if your features are not well distributed over the image frame, if you do not have enough features or if your descriptors are not distinctive enough to generate reliable matches then RANSAC may fail. This means you may not have enough information to reliably compute the transformation between the frames. Second, by using this kind of feature-based method, we are ignoring a lot of additional information that is present within pairs of RGB D frames. Recall from the sensor modality chapter the quantity of data that we get from an RGB D frame; selecting even 1000 features is only a small fraction of the total amount of depth and color data that is present in an RGB-D image. This provides the motivation behind merging structure and texture information when computing frame to frame alignment.

Assume, for example, that we are looking at a scene with very few image features. In this case, features are intensity gradient configurations that trigger a particular

feature detector. This scene contains a set of boxes with uniform color, placed in the middle of the room with sufficient ambient lighting such that it is hard to obtain more than a few 10s of features (similar to one of the Freiburg RGB-D datasets [195]). Even still, those features may not be particularly distinctive enough to create reliable matches between neighboring RGB-D frames. The structured light sensor, however, would be generating a depth map that could otherwise provide enough information to determine the relative poses between 2 neighboring RGB-D frames using an algorithm such as ICP to align the dense point clouds from each frame.

A third problem with feature-based visual odometry is that feature extraction is often too expensive to run at a camera's frame rate, especially for the more reliable features and descriptors (e.g., SIFT [132]). If we avoid computing feature descriptors in the typical case, then we must make use of as much of the raw information as possible, leading to the principles of the dense methods.

5.3 Methods for Dense Alignment

In the following sections, we discuss three methods for dense alignment appropriate for RGB-D sensor use: ICP, DVO, and spherical harmonics. ICP specifically utilizes point cloud information to iteratively minimize the distance between closest points, while DVO warps an intensity image (i.e., the grayscale conversion of the RGB data) based on depth in order to minimize the photometric error. Both methods are described below, followed by the method we used to compute the joint model. Finally we discuss the less well-known model of spherical harmonics and how it can be used for ego-motion estimation.

5.4 ICP

In 1992 Besl and McKay describe a method for registering two shapes based on a function that, given a point on one shape, computes the closest point on the other shape [13]. The closest point correspondences are used to compute a transforma-

tion between the query shape that moves the shapes into better registration; better registration is defined as minimizing the squared distance between the closest point correspondences. However, since a single iteration can change the closest point correspondences, this is done iteratively until a termination criterion has been met. They called this method the ICP algorithm. ICP will converge to the nearest local minimum that optimizes a mean-squared distance metric between the point sets. While Besl and McKay formulate the approach in a generic framework to handle many different geometric representations, we focus exclusively on the 3D point cloud formulation described next.

Given two clouds $\mathcal{P}_1$ and $\mathcal{P}_2$ of points in $\mathbb{R}^3$ and some distance metric $d : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$, finding the closest point $p_j \in \mathcal{P}_2$ for each point $p_i \in \mathcal{P}_1$ allows one to compute the transform $T_{\mathcal{P}_1 \rightarrow \mathcal{P}_2}$ to minimize the distance between each pair of corresponding points. We define

$$T_{\mathcal{P}_1 \rightarrow \mathcal{P}_2} = [\mathbf{R} \mid \mathbf{t}], \tag{5.1}$$

where $\mathbf{R} \in SO(3)$ and is represented by a 3×3 rotation matrix and $\mathbf{t}$ is a 3×1 translation vector. Besl and McKay's algorithm then iteratively minimizes the following energy function:

$$\underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} E(\mathcal{P}_1, \mathcal{P}_2) = \sum_{p_i \in \mathcal{P}_1, p_j \in \mathcal{P}_2} \left[d(R\mathbf{p}_j + \mathbf{t}, \mathbf{p}_i) \right]^2, \tag{5.2}$$

where d is the Euclidean distance function between points.

However, unlike the goals for a typical feature matching algorithm, the corresponding nearest points usually are **not** the actual point that would best match between the clouds, i.e., they may not be points that would correspond given a globally optimal transform. There is also a difference between aligning two clouds that are (almost) exactly the same and just vary by their pose, and matching two clouds of a scene that are generated from one or more sensors under different viewpoints. In the first case, there is an "exact" correspondence between points in cloud $\mathcal{P}_1$ and cloud $\mathcal{P}_2$. In the second case, there is likely no exact correspondence, which means that any transform between point correspondences will ultimately contribute some error to the overall

cloud transformation estimate. In addition, since the observation of the environment from an RGB-D camera is inherently a discrete, noisy sampling of the surfaces in view then it is highly unlikely that two corresponding points will match exactly. Is there another, more effective way to compute the distance between the point clouds' implicit surfaces?

Chen and Medioni described a similar method they developed for registering range images for the purpose of constructing a 3D representation of an object [27]. They minimize a very similar energy function, but instead of point to point distances, they minimize the distance between a query point $\mathbf{p}_j$ and the tangent plane $\nu(\mathbf{p}_i)$ of the model point $\mathbf{p}_i$ (defined by the point and its normal $\nu(\mathbf{p}_i)$):

$$\underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} E(\mathcal{P}_1, \mathcal{P}_2) = \sum_{p_i \in \mathcal{P}_1, p_j \in \mathcal{P}_2} \left[((R\mathbf{p}_j + \mathbf{t}) - \mathbf{p}_i) \cdot \nu(\mathbf{p}_i) \right]^2 . \tag{5.3}$$

Intuitively, using the point to plane metric makes sense when considering two different point clouds (since point locations will likely not match but the implicit surfaces and their normal vectors should), and it has also been shown to converge more quickly and reliably, especially when the surfaces are already nearby [170]. Therefore, we choose to align two RGB-D point clouds from a moving sensor using the point to plane function to compute the incremental ego-motion transformation.

Note that this does require an extra step to process the point cloud to generate normal estimates at every point. Section 5.4 discusses how to do this.

A detailed account of ICP is not the purpose of this section; we refer the interested reader to the reviews by Pomerleau, Colas and Siegwart [168] and Wang and Zhao [209]. Instead, we highlight the important aspects of the algorithm relevant to aligning nearby frames from a frame-rate RGB-D camera.

Algorithm initialization

Preparing an RGB-D frame for ICP requires several steps. First, we utilize a coarse-to-fine processing approach to help handle larger transformations between camera frames. We accomplish this by computing a pyramid of RGB and depth images. The

original images are blurred and subsampled to generate the next layer in the pyramid, and this is repeated until the desired number of levels are generated or the resolution is smaller than some threshold. In this work, we use 3 additional levels (a total of 4 levels when considering the original resolution) that yields pyramid image sizes of $(80, 60), (160, 120), (320, 240), (640, 480)$.

Second, since raw depth data is often noisy, even for near-depth scenes, it is helpful to reduce the noise through a filtering operation to help improve the normal generation and point-plane matching. While this may eliminate smaller details in the depth map and effectively blur small depth transitions, the details tend to be less important for visual odometry since the larger surfaces on average are contributing more to the alignment than many of the small details[1]. We do this through an algorithm called bilateral filtering [200] applied to the depth image. This process smooths depth values within a specified metric radius and depth threshold to avoid considering values that are too far apart and artificially blurring sharp edges and depth discontinuities.

Finally, we generate normals for each point in the point cloud. This procedure is described in the next section.

Normal computation

There is no sensor that can directly provide surface normals from the environment; but surface normals are a useful way of summarizing the characteristics of a surface at a given point, and they are integral for computing the point to plane metric in ICP. Therefore, we approximate surface normals at a point by examining its neighbors within the organized structure of the depth image. There are several different ways to compute the neighbors, as well as several different ways of computing the normal given the neighbors. We describe the two methods readily available to us, and why we use one over the other.

Computing the neighbors To find a point's neighbors in a general, *unorganized* point cloud, one must utilize a spatial data structure. Given a set of points that have

[1] Although small details *can* be important in some situations, see [71].

been processed into such a structure, one then has access to an efficient interface for querying points within the structure with respect to several spatial attributes. For example, one may wish to search for points that are nearby (i.e., are neighbors) of a provided query point; this is usually accomplished by finding a set of k nearest neighbors (and their distances) or by finding *all* points within a specified radius r of query.

Two typical spatial data structures in use for point clouds (and provided by Point Cloud Library (PCL)) are octrees and k-d trees. Octrees are an extension of the 2D quadtree spatial data structure to 3 dimensions. In an octree, the total bounding box of all points is considered and then split into 8 octants (imagine dividing a cube into 8 equal parts by dividing each face into four equal squares). This first split becomes the root of the tree. Then, for any occupied octant, another subtree is added by recursively subdividing that octant, the recursion terminates when a depth threshold is reached or no more points are found in any of the leaf nodes. If every point is in a separate leaf, and the octree has depth d, then it takes no more than d operations to find a point, and on average $\log_8(n)$ operations where the cloud has n points. Unfortunately, d may be larger than $\log_8(n)$, since the depth depends on the structure of the points in space; if many points are clustered tightly and each leaf must contain no more than 1 point, then, like an unbalanced binary tree, the depth may be worst-case $O(n)$.

The other data structure, k-d trees, are meant to better account for this situation, while still handling the tightly clustered case mentioned above [11]. In addition, a k-d tree is able to handle dimensions higher than 3 by generalizing a binary tree to k dimensions. It does this by recursively dividing the point set using the succession of dimensions (e.g., for a cloud in $\mathbb{R}^3$, $\{1, 2, 3, 1, 2, 3, 1, 2 \ldots\}$), and then attempting to ensure that the number of points in each subtree are relatively balanced by splitting on the median of the values for the selected dimension. This is only possible when the points are known in advance; when points are inserted dynamically, there is often nothing that can be done to guarantee the tree will be balanced. Instead, the client can decide how much balance is required, and choose to periodically rebalance when

the tree becomes sufficiently unbalanced.

In contrast, when a point cloud is *organized*, as in the case of the depth maps provided by typical RGB-D cameras, looking up neighbors is much more directly accomplished by simply indexing the points by the neighboring image coordinate values. While this provides the ability to find potential neighbors, it still requires looking at the distance between the points, since if it is greater than the target radius, it should not be considered as a neighbor.

Computing the normal A simple, geometric way to compute the normal is by finding the cross product of two vectors defined by the target point and two neighbors. The three points determine a triangle, and the cross product provides the normal direction of the triangle plane. Using only three points will likely produce a very noisy normal map; therefore, a slightly more robust approach is to compute the mean of all normals in the neighborhood.

A better and more prevalent approach, but also more expensive, is to find the best plane that fits the neighborhood where the squared distance from each point to the plane is minimized. Hoppe describes this in his paper on surface reconstruction [95].

Therefore, we estimate a normal at point p_i using the following approach:

- find the points $\{p_j : p_j \in \mathcal{N}_{\mathbf{p_i}}\}$ in some neighborhood of p_i
- compute the covariance matrix, where p_i is assumed to be the centroid of the neighborhood
- perform principal component analysis of the covariance matrix, finding the eigenvector $\mathbf{e}_0$ corresponding to the smallest eigenvalue
- $\mathbf{n}_i = \frac{\mathbf{e}_0}{\|\mathbf{e}_0\|}$

Computing normals is a challenging problem in itself and may involve tradeoffs and specialized computations to handle the general case [178] (e.g., computing the neighborhood size adaptively based on the density of points in the local neighborhood). For an RGB-D camera with an "organized" point cloud, however, it is possible

to compute estimated normals quickly and efficiently[2], and this operation can be done in parallel.

Normals can be used in a variety of computations, not just for ICP... For example: segmentation (i.e. oversegmentation in Chapter 7, LCCP [31]) as well as rotation estimation for cloud alignment using spherical harmonics (in the computation of the extended Gaussian image, see section 5.7).

Finding Correspondences

The nature of the ICP algorithm requires correspondences between points on every iteration. Correspondences are found intentionally in the basic feature-matching algorithm, but in this case, we are not using features, and we'd like to avoid expensive spatial data structure creation and lookup for each point in the query frame. We solve this problem by making the assumption that when operating at frame rate, the resulting clouds will be relatively close together[3], and therefore it is feasible to project the query points into the previous frame (initially with an identity transform) to find the correspondences. This was first demonstrated in the work on Kinect Fusion [99], [152].

We use the estimated model to query transformation in each iteration to compute the corresponding point pairs and to compute the Jacobian used in the optimization:

- transform the model points into the query frame
- project the model points onto the depth image plane, discarding points outside the viewing frustum
- for each projected point p_i, compute the interpolated query point q_j

[2]Although perhaps not as accurately as possible.

[3]Note that this assumption does not always hold, especially for commercial RGB-D cameras with "low" framerates and faster motions; therefore for most of these algorithms, the robot or user must move the camera slowly. While not a very practical restriction, this can be considered a hardware problem; it could be addressed with higher framerate global shutter RGB-D cameras. One may imagine that even these could hit motion limitations, and therefore research that investigates effective low-cost dense correspondence algorithms would be relevant.

Optimization

ICP is formulated as an iterative non-linear least squares problem for two reasons: the *true* correspondences are not known ahead of time, and the transformation of points involves non-linear functions of the rotation parameters. In practice, we linearize around the identity at each iteration and formulate the problem as a linear least squares problem.

Using the point-plane construct, the initial problem is formulated as:

$$\mathbf{E}_{\text{icp}} = \sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \hat{\xi}\mathbf{T}\mathbf{p}_n^{t+1})^\intercal \cdot \mathbf{n}^t \right]^2 \tag{5.4}$$

where $\mathbf{T}$ is the current estimate of the transform, $\mathbf{p}_n^{t+1}$ the current point in the query frame, $\mathbf{p}^t$ and $\mathbf{n}^t$ are the corresponding point and normal in the model, and

$$\hat{\xi} = \begin{bmatrix} [\omega]_\times & \mathbf{v} \\ \mathbf{0} & 0 \end{bmatrix} \tag{5.5}$$

is an element of the Lie algebra $\mathfrak{se}(3)$, representing the differential transformation we are estimating in this iteration. With some algebraic manipulations after linearizing around the identity, we get the resulting Eq. 5.14.

$$\mathbf{E}_{\text{icp}} = \sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - (I + \hat{\xi}) \mathbf{T} \mathbf{p}_n^{t+1}) \cdot \mathbf{n}^t \right]^2 \tag{5.6}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \mathbf{T} \mathbf{p}_n^{t+1} - \hat{\xi} (\mathbf{T} \mathbf{p}_n^{t+1})) \cdot \mathbf{n}^t \right]^2 \tag{5.7}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \mathbf{T} \mathbf{p}_n^{t+1}) \cdot n^k - (\hat{\xi} \mathbf{T} \mathbf{p}_n^{t+1}) \cdot \mathbf{n}^t \right]^2 \tag{5.8}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \mathbf{T} \mathbf{p}_n^{t+1}) \cdot n^k - \left(\begin{bmatrix} [\omega]_\times & \mathbf{v} \\ \mathbf{0} & 0 \end{bmatrix} \mathbf{T} \mathbf{p}_n^{t+1} \right) \cdot \mathbf{n}^t \right]^2 \tag{5.9}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \mathbf{T} \mathbf{p}_n^{t+1}) \cdot n^k - ([\omega]_\times \mathbf{T} \mathbf{p}_n^{t+1} + \mathbf{v}) \cdot \mathbf{n}^t \right]^2 \tag{5.10}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \mathbf{T} \mathbf{p}_n^{t+1}) \cdot n^k - (\omega \times \mathbf{T} \mathbf{p}_n^{t+1} + \mathbf{v}) \cdot \mathbf{n}^t \right]^2 \tag{5.11}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \mathbf{T} \mathbf{p}_n^{t+1}) \cdot n^k - \begin{bmatrix} \mathbf{T} \mathbf{p}_n^{t+1} \times \mathbf{n}^t \\ \mathbf{n}^t \end{bmatrix}^\mathsf{T} \begin{bmatrix} \omega \\ \mathbf{v} \end{bmatrix} \right]^2 \tag{5.12}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} \left[(\mathbf{p}^t - \mathbf{T} \mathbf{p}_n^{t+1}) \cdot n^k - \begin{bmatrix} \mathbf{T} \mathbf{p}_n^{t+1} \times \mathbf{n}^t \\ \mathbf{n}^t \end{bmatrix}^\mathsf{T} \xi \right]^2 \tag{5.13}$$

$$\sum_{\mathbf{p}_n^{t+1} \in I_{t+1}} [-\mathbf{J}_{\text{icp}} \xi + \mathbf{r}_{\text{icp}}]^2 \tag{5.14}$$

5.5 Dense Visual Odometry

While ICP focuses on aligning point clouds by minimizing the point to plane distance between all point correspondences, Steinbrucker's dense visual odometry algorithm focuses on minimizing the photometric error between corresponding points[107], [192]. The algorithm is predicated on the photometric constancy assumption, i.e., that given two nearby but different viewpoints of the same scene, a single point on a surface reflects the same amount of light and therefore produces a similar intensity for the

corresponding point in each view[4]. Whether this is a realistic assumption will depend on the nature of the scene, including the lighting, surface materials, and the nature of the transformation between the two views. Practically, and ignoring prevalent but minimal specular effects, smaller camera motions support the assumption better. The camera itself is also a complicating factor: most RGB-D cameras provide an automatic exposure and automatic white balance function that can significantly change the overall brightness of the scene and otherwise confuse the photometric function since corresponding points no longer have the same intensity value between frames (more recent[5] direct VO methods take great pains to account for this effect by calibrating and estimating online exposure parameters, for example, see [12], [50], [234]).

Why is this approach interesting, when the assumption is so tenous? Ultimately, an ICP approach will only function on point clouds with enough geometry to constrain the least squares optimization. DVO, on the other hand, is capable of using geometry *and* appearance to compute the information needed to align frames. While this may not be as effective at aligning individual pixels as using point features derived from appearance (i.e., indirect methods with keypoints and descriptors), it is the dense information over large regions of the image that allows the algorithm to compute a transform even when there is no significant geometry; e.g., when looking at a poster on a wall.

Dense visual odometry (DVO) exploits the depth / intensity pair produced by RGB-D cameras, and operates using the following equation, where a point on a surface is assumed to have the same appearance (i.e., intensity) when seen from two different viewpoints:

$$I_t(\mathbf{x}) = I_{t+1}(\tau(\mathbf{x}, \theta)), \tag{5.15}$$

where $\mathrm{I_t}$ and $\mathrm{I_{t+1}}$ are consecutive images, $\theta \in SE(3)$ and $\tau(\mathbf{x}, \theta)$ is a warping function that maps pixels from frame t to $t+1$ such that the error derived from Eq. 5.15 is

[4]I would be inclined to call this the photometric constancy *simplification*, since there is rarely a case where this would actually hold.

[5]At least, more recent than when this work was performed.

minimized:

$$\mathbf{E}_{\text{dvo}}(\theta) = \int_{\mathbf{x}\in\Omega} \left[I_{t+1}(\tau(\mathbf{x},\theta)) - I_t(\mathbf{x})\right]^2 d\mathbf{x}. \tag{5.16}$$

The warping function, in particular, enables us to compare frames by transforming the points in frame $t+1$ to frame t using the currently estimated transformation parameters, and subsequently projecting them to the image plane to determine their intensity values. The DVO algorithm proceeds by iteratively computing the transformation that minimizes the $\mathbf{E}_{\text{dvo}}$ energy. Each iteration computes the Jacobians based on mapping the current frame intensity image onto the transformed point cloud from the previous frame, then computes the difference between the expected intensity values and the mapped intensity values. A set of partial derivatives are computed at each point, along with the residual, and each is accumulated into the normal equations for the least squares optimization.

The DVO energy is optimized using an iteratively reweighted least squares approach in an inverse compositional formulation. The original formulation is the standard forward additive procedure proposed by Lucas and Kanade in their optical flow paper [133]. The objective function is optimized with respect to the change in the parameters of the warping function ($\theta \in SE(3)$) and the parameters are updated by simply adding the delta values. In the inverse compositional model [5], the objective is modified to estimate the incremental warp (the composition of warps, $\theta = \Delta\theta + \theta$) as opposed to just the change in the parameters. While this is provably equivalent to the forward additive method if the set of warp functions form a group, there is a computational benefit: the Hessian may be computed once, as it becomes independent of the parameters. For more details, we refer the reader to the overview paper by Baker and Matthews [5].

To handle larger transformations between frames, particularly due to the photometric constancy assumption, DVO uses a coarse to fine approach. An image pyramid is generated using the pyramid down operator (Gaussian filter followed by a subsampling). Then, starting from the lowest resolution of the pyramid, the algorithm is run to determine the estimated transformation parameters for that level, followed

by the optimization at the next level using an initialization from the previous level parameters; this continues until the highest resolution pyramid level is used.

In practice, we found that when the photometric constancy assumption was violated or an image difference was too large that the algorithm diverged. Since DVO, like ICP, is based on a non-linear least squares optimization approach, each step uses a linearization of the gradient. Therefore, DVO is a local optimization, and may not converge to a globally optimal solution.

5.6 Joint ICP-DVO

For ICP, *not* using feature descriptors derived from appearance means we are relying on the nearness of the point cloud and its geometric structure to provide effective cues for alignment. However, this is a often a faulty assumption. Either the camera may move too quickly to make this assumption (i.e., the difference between the camera poses is too large to support the projective data association), or perhaps there's too much noise in the depth image, or the point cloud geometry is not distinct enough to provide constraints for performing alignment. For example, while an RGB camera can effectively compute the points that lie on the surface of a wall, the wall has no geometric features that allow us to align it with any other version of a wall; if we take two snapshots of a wall, translationally moving in any direction parallel with the wall, then there are likely no distinct geometric features to constrain the position between the two frames. We can align the planes but then we still have three additional degrees of freedom: rotation around the normal of the plane, as well as the x, y position in the plane.

Dense visual odometry on the other hand, can make use of any visual features that may reside on the surface of the wall as long as they generate a detectable gradient in the image. For example, if there is a poster mounted on the wall, then it is likely there are enough gradients in the image to allow us to generate the three additional constraints on top of the plane alignment. Unfortunately, dense visual imagery comes with its own set of challenges. For example, the photometric constancy assumption

is often broken with cameras moving dynamically in a scene given just the effects of specular reflection and shadows that may appear and disappear depending on light sources within the environment, not to mention the dynamic white balance and exposure algorithms enabled by default on many cameras (as well as the camera we used). Imagine the DVO algorithm seeing the same image, possibly with very little motion, but with a significantly different exposure that changes the observed intensity at many points; instead of converging to the correct minimum, it will likely diverge; this brings us to the idea of combining the two algorithms.

One of the central ideas of our work in this chapter is to combine the benefits of geometry-based alignment with the benefits of appearance-based alignment. Since both algorithms involve estimating the incremental transform between pairs of images, we just need to formulate the optimization as the sum of the energy functionals, where λ is a weight to balance the energy terms:

$$\mathbf{E}_{\text{joint}} = \mathbf{E}_{\text{dvo}} + \lambda \mathbf{E}_{\text{icp}}. \tag{5.17}$$

If $\mathbf{J}_{\text{icp}} \in \mathbb{R}^{n \times 6}$, $\mathbf{J}_{\text{dvo}} \in \mathbb{R}^{m \times 6}$, then

$$\mathbf{J}_{\text{joint}} \in \mathbb{R}^{(n+m) \times 6} = \begin{bmatrix} \mathbf{J}_{\text{dvo}} \\ \lambda \mathbf{J}_{\text{icp}} \end{bmatrix}, \tag{5.18}$$

and we iteratively solve the following least squares normal equation for $\Delta\theta$:

$$\mathbf{J}_{\text{joint}}^{\mathsf{T}} \mathbf{J}_{\text{joint}} \delta\theta = \mathbf{r}. \tag{5.19}$$

5.6.1 Incremental versus keyframes

When one wishes to compute the incremental transformation between frames over time, there are several ways it can be done. Two obvious possibilities are: (1) compute the incremental transformation between frame F_i and F_{i+1} as T_i^{i+1} and then append

to the global transformation as:

$$T_0^{i+1} = T_i^{i+1} T_0^i, \tag{5.20}$$

or, alternatively, (2) compute a transformation relative to a keyframe, and only append to the global transformation when the keyframe changes. Keyframes are a well-known technique in visual odometry (derived from the process of keyframing in animation) to help reduce error by minimizing the number of compositions performed using a stable model source. A keyframe is a frame selected to be the model for some time period, usually subject to some kind of validity test. In the case of visual odometry, the keyframe may remain valid while the camera's frustum overlaps a sufficient portion of the keyframe frustum. Once this constraint is violated, a new keyframe is selected (i.e., the current active frame) and the global transformation of the new keyframe is computed as in 5.20 but where i represents the index of the keyframes, a set often significantly smaller than all the original frames.

In many algorithms a small set of possibly overlapping or nearby keyframes remains active in order to better compute a joint transformation (e.g., the active keyframe set in [46]). In our implementation of the joint algorithm, we have found that the keyframe method performs better that frame to frame accumulation.

5.6.2 GPU Acceleration

It is expensive to run this algorithm sequentially on a single processor. While there are many ways to speed an algorithm up, splitting it into subtasks that can run concurrently and then running them in parallel on multiple processors is often the most effective. This does assume that you are already using the most efficient algorithm for the task, but sometimes less efficient algorithms that can be split into concurrent subalgorithms can be more efficient than the more efficient sequential algorithm. This is a complex topic, and not necessarily within the scope of this section. Suffice it to say, there are several aspects of the ICP and DVO algorithm that can be run efficiently in parallel, especially when considering the capabilities of modern GPUs.

There are many computations of DVO and ICP that can be performed in an embarrassingly parallel fashion. For each point, we can independently find its correspondence and independently compute the distance and the point to plane algorithm. Likewise, for dense visual odometry, we can warp the pixels in parallel, i.e., compute the image derived from the point cloud with the image intensities, then find the correspondences and compute the gradients independently per point. This makes these algorithms prime candidates for parallelization and a massively parallel architecture such as NVIDIA's CUDA framework.

Briefly, NVIDIA's CUDA framework is a general purpose GPU programming language specifically designed to take advantage of the parallel hardware in GPUs. However, each one of these little units is a relatively general computing processor, and they operate in parallel. In fact, they operate in a single instruction multiple thread (SIMT) framework where multiple threads operate in lock-step on a fixed set of instructions. This set of instructions is called a kernel, and it is a program for single parallel threads and describes how they operate on one or more independent data. Depending on how you have partitioned the data, the kernel operates on those partitions in parallel based on the number of processing cores available in your hardware device. Whereas a desktop or laptop computer may have anywhere from 4 to 64 cores, a GPU can have 768 to over 3000 cores. Granted, each core runs a more slowly and is not as powerful or complex as a CPU core, but the primary benefit is in their massive parallelism.

One of the biggest challenges in implementing the optimization algorithm for joint DVO and ICP using the CUDA architecture was performing the reduction algorithms on the GPU. While it is possible to implement the reduction algorithm on the CPU, there are limits of host/device memory transfer, so that it is relatively expensive to transfer data back and forth from the GPU to CPU. If we are determined to keep both data and computation on the GPU, then we must handle the reduction algorithms that are not inherently parallel. However, we can take advantage of as much parallelism as possible.

What is a reduction algorithm? If you are familiar with map/reduce algorithms

in distributed, parallel processing, then we can describe the entire set of GPU operations for the joint DVO/ICP least squares algorithm as a map/reduce operation. Specifically, we *map* the error computation to every single point in the query frame, and then we *reduce* by combining all of the individual errors into a global error data structure: the normal matrix (see Eq. 5.19). In general, the reduction phase cannot be done completely in parallel, because part of the task is combining the values from several individual data. Given the shared memory constraints of the parallel CUDA hardware, we implemented a staged reduction algorithm to best exploit the local cache of the GPU in order speed up the reduction operation.

The joint optimization framework is similar to the work of Whelen et al. in their Kintinuous approach [217], with several differences. We use a surfel-based representation instead of TSDF (described in detail in the next chapter), and we optimize the DVO computation on the GPU to avoid the data transfer.

In the following sections, we describe the methodology for converting the algorithm described above into kernels for the CUDA parallel processing framework running on NVIDIA GPUs.

Image Processing

Many image processing algorithms can be carried out quickly on the GPU since they often operate in the neighborhood of a pixel and are therefore amenable to parallelization at the pixel level. Since we need to perform several operations on the incoming RGB-D image frames, therefore producing several images, it makes sense to transfer the original images to the GPU and do the operations there; they will be faster to compute and require fewer memory transfers between the CPU and GPU.

We implement the following image processes for use by the ICP and DVO portions of the joint alignment algorithm:

- bilateral filtering (for depth image),
- normals (for ICP),
- pyramid downsampling (for DVO), and

- horizontal and vertical derivatives (for DVO).

The core (or kernel) of the algorithms are the same whether on CPU or GPU, but whereas the CPU algorithm will have the kernel operate within one or more loops to process each pixel sequentially[6], the GPU algorithm is expressed as a kernel that runs on a single GPU core on the device, and a host-based function launches the kernel with a specification indicating the requested parallelism for the kernel. This specification includes information on the number of blocks and number of threads per block, as well as how to spatially distribute the threads. For example, if you are processing a 1 dimensional array of 100 values, then you may want to generate 10 blocks of 10 threads each, and if the GPU has the resources, each thread in each block will run in parallel. The kernel itself is then run 100 times, and each time it receives (in special variables) the block index and the thread index for that specific kernel execution. Similarly, if you are processing a 2 dimensional array (e.g., an image), then you can specify both the number of blocks in x, y and the number of x, y threads as well. The benefit of processing images using 2D spatially organized blocks of threads is more coherent memory access for algorithms that need to process neighboring pixels.

ICP Computation

We provide pseudocode for the ICP least squares reduction kernel in Algorithm 2. The kernel is executed for every valid point in the model keyframe that falls in the query frame frustum, and computes the contribution of the point in the least squares energy functional. During our experimentation, we discovered a small bottleneck in the GPU reduction: global memory access was slowing things down for each frame. Line 12 indicates our solution to this problem: after initializing the largest shared memory region possible for our level of block parallelism[7] in line 3, each thread has access

[6]CPU parallelism is also possible, and is easily done within loops using a compiler extension such as OpenMP. However, the maximum number of usable threads (and therefore the level of parallelism) is proportional to the number of cores; while some popular CPUs may have between 4 to 8 cores, the number of (simpler) cores on a GPU typically numbers somewhere in the thousands.

[7]In CUDA programming on NVIDIA GPUs, the program can reserve the use of an amount of shared memory that will be available to all threads in a block. Shared memory is a very limited resource, but can be accessed around 100 times faster than local or global memory on the GPU.

Algorithm 2 On-GPU ICP kernel implementation

1: **function** ICP_LS_REDUCE($blocks, model, query, T$)
2: initialize local variables
3: $D_s \leftarrow$ init_kernel_shared_data()
4: $x, y \leftarrow$ kernel_coords()
5: $p, n \leftarrow$model(x, y) ▷ Get the point and normal from the model given x, y
6: $p', n' \leftarrow \hat{T}(p, n)$ ▷ Transform the model point to query frame
7: $u, v \leftarrow \pi(p')$ ▷ Project to find the query image coords
8: $n_q, d_q \leftarrow$ **interpolate**$(query, u, v)$ ▷ Interpolate query image data
9: $p_q \leftarrow$ **backproject**(u, v, d_q)
10: $\mathbf{J}_{\text{icp}} \leftarrow \begin{bmatrix} -n_q \\ p' \times n_q \end{bmatrix}$
11: $r_{\text{icp}} \leftarrow (p_q - p') \cdot n_q$
12: **for** $i \leftarrow 0$ to 4 **do** ▷ Phased normal eq. reduction over valid Jacobians
13: $D_s \leftarrow$ **normal_eq**$_{6i\ldots(6i+5)}(\mathbf{J}_{\text{icp}}, r)$
14: $blocks_{6i\ldots(6i+5)} \leftarrow$ **reduce**(D_s)
15: **end for**
16: **end function**

to 24 bytes of ultra-fast memory. This allowed us to implement a phased approach to the least squares reduction, where in each phase up to 6 elements are computed and reduced across all participating threads. By using the shared memory for the reduction, we can avoid costly global memory accesses and improve the reduction runtime.

DVO Computation

The DVO computation is slightly more complex than the ICP process. Whereas the majority of the ICP algorithm can be implemented in a single kernel (as shown in algorithm 2), the DVO algorithm is spread over multiple kernels to better represent the logical distinction between the operations. Therefore, when describing this algorithm, we describe the high-level operations (implemented on the host) and only show the implementation of the DVO least squares reduction. The latter is very similar to the ICP reduction, but requires different Jacobian computations (see lines 5-10). Note that every operation in algorithm 3 occurs in parallel on the GPU, including the

In this case, due to our computed block level parallelism (i.e., how many blocks we request for the kernel), we could reserve enough space for 6 floats *per thread*.

Algorithm 3 Host-based DVO computation

function COMPUTE_DVO($blocks, model, query, T$)
  $\mathcal{V} \leftarrow$ **vertex_map**$(model)$
  $\mathcal{I}_w^q \leftarrow$ **warp**$(T, \mathcal{V}, query)$
  $d_x, d_y \leftarrow \left\{ \frac{d\mathcal{I}^m}{dx}, \frac{d\mathcal{I}^m}{dy} \right\}$
  $d'_x, d'_y \leftarrow \{f_x d_x, f_y d_y\}$
  $\mathbf{r}_{\text{dvo}} \leftarrow \mathcal{I}_w^q - \mathcal{I}^m$
  $\sigma \leftarrow$ **est_scale**$(\mathbf{r}_{\text{dvo}})$
  $\mathbf{w} \leftarrow$ **tdist**$(\nu, \sigma, \mathbf{r}_{\text{dvo}})$
  dvo_ls_reduce$(blocks, \mathcal{V}, \mathbf{r}_{\text{dvo}}, \mathbf{w}, d'_x, d'_y)$
end function

Algorithm 4 On-GPU DVO kernel implementation

function DVO_LS_REDUCE($blocks, \mathcal{V}, \mathbf{r}_{\text{dvo}}, \mathbf{w}, d'_x, d'_y$)
  $D_s \leftarrow$ init_kernel_shared_data()
  $x, y \leftarrow$ kernel_coords()
  $v_x, v_y, v_z \leftarrow \mathcal{V}_{x,y}$
  $J_0 \leftarrow v_z d'_x$
  $J_1 \leftarrow v_z d'_y$
  $J_2 \leftarrow d'_x(-v_x v_z^2) + d'_y(-v_y v_z^2)$
  $J_3 \leftarrow d'_x(-v_x v_z^2 v_y) + d'_y(-1 + -v_y v_z^2 v_y)$
  $J_4 \leftarrow d'_x(1 + v_x v_z^2 v_x) + d'_y(v_x v_z^2 v_y)$
  $J_5 \leftarrow d'_x(-v_y v_z) + d'_y(v_x v_z)$
  $\mathbf{J}_{\text{dvo}} \leftarrow J_{0\ldots5}$
  for $i \leftarrow 0$ to 4 **do** ▷ Phased normal eq. reduction over valid Jacobians
    $D_s \leftarrow$ **normal_eq**$_{6i\ldots(6i+5)}(\mathbf{J}_{\text{dvo}}, r)$
    $blocks_{6i\ldots(6i+5)} \leftarrow$ **reduce**(D_s)
  end for
end function

computation of the vertex map from the depth image (**vertex_map**), the vertex map warping (**warp**), the error scale estimation (**est_scale**), and the Student's t distribution computation for generating weights (**tdist**).

5.7 Spherical Harmonics

In this section, we discuss the use of spherical harmonics as another approach to dense alignment appropriate for RGB-D sensors. Spherical harmonics is a way to represent functions on a sphere as the integral of a set of basis functions. They allow

a generalization of Fourier analysis of periodic functions to functions defined on the sphere, analogous to the use of circular functions as a basis for periodic functions on a circle. An interesting aspect of Fourier analysis is that two periodic functions on a sphere, when decomposed into their corresponding basis functions, have identical frequency distributions if they are the same modulus a rotation. This leads to the natural idea of considering spherical harmonics when representing and decomposing a function of an environment as seen from different views of a camera. In other words, if we can find *some* way to represent a static environment as a spherical function, then it may be possible to compute the rotation between two different (but overlapping) views of the same environment[8].

This section describes the use of spherical harmonic functions derived from extended Gaussian images (EGIs) for online egomotion estimation, and compares the performance to other methods, including an ego-motion estimation algorithm based on an implementation of the frame to frame skeleton in Algorithma 1. An EGI provides a rotation-equivariant representation of dense depth maps that is completely independent of appearance information and requires no explicitly computed sparse features [96]. The implications are significant: like DVO, no explicit features need to be extracted, but unlike DVO (and more like ICP), the rotation estimate only depends on the normals in the structure of the environment.

The primary contributions are: 1) an implementation of online correspondence-less egomotion estimation using spherical harmonic analysis, and 2) a comparison of egomotion estimation methods using synchronized ground truth data. We use two data sets for testing: one collected at our lab, the other available in publicly [196]. All algorithms are implemented using the Robot Operating System (ROS) [173], with VO algorithms using OpenCV [18].

[8]The material in this section is adapted from [156], and is reprinted with permission of the first author (©2012 IEEE). The spherical harmonics implementation and evaluation was done by Philip Osteen, while I contributed the feature-based visual odometry implementation and some of the analysis. This work was completed in 2011, chronologically *before* the joint DVO/ICP work.

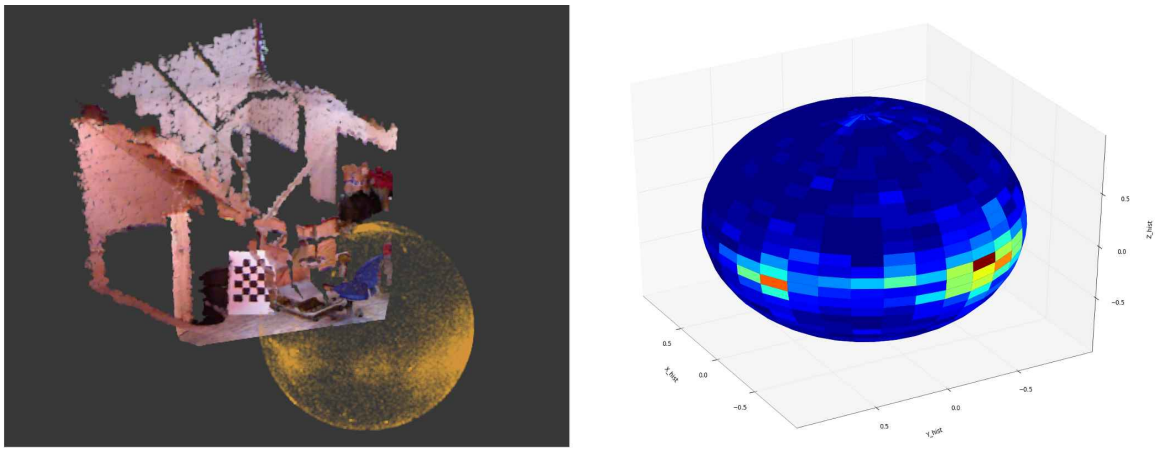


Figure 5.1 At left, an RGB-D point cloud and its associated EGI. At right, the 2D histogram is formed by discretizing the EGI, highlighting the distribution of local normals. The correlation of consecutive EGIs yields a 3 DoF rotation estimate. Reproduced from [156].

5.7.1 Method

Our process begins by representing a point cloud with its associated EGI. The EGI is created by estimating the local normal at each point in a point cloud, using tools from the Point Cloud Library (PCL) [180]. The normal direction at a point is estimated as in §5.4. For our purposes, the EGI can be interpreted as a sampling of a function defined on the unit sphere. Next, we discretize the sphere, and the EGI is converted into a 2D histogram. Our work employs the equiangular discretization, as employed in [84], to tesselate the sphere. The colatitude $0 \leq \theta \leq \pi$, measured down from the z-axis (north pole), and longitude $0 \leq \phi \leq 2\pi$, measured from the x-axis, are the angles used to discretize the sphere. Figure 5.1 illustrates the process of creating a 2D histogram from an input point cloud, as well as the corresponding EGI.

We use the fact that the rotation $\mathbf{R} \in SO(3)$ between functions on the unit sphere can be directly determined by correlating two functions f and h and applying an $SO(3)$ Fourier Transform (SOFT) to the result.

From [115], the correlation of spherical functions f and h is itself a function defined on $L^2(SO(3))$, and the maximum value of the correlation will yield the desired rotation g. The correlation can be represented as

$$C(g) = \sum_{l \geq 0} \sum_{m=-l}^{l} \sum_{m'=-l}^{l} \hat{f}_m^l \, \overline{\hat{h}_{m'}^l} \, \overline{D_{mm'}^l(g)}, \tag{5.21}$$

with $\overline{D_{mm'}^l}$ representing the complex conjugate of $D_{mm'}^l$. The functions $f, h \in L^2(SO(3))$ are decomposable using Fourier analysis, as:

$$f(\alpha, \beta, \gamma) = \sum_{J \geq 0} \sum_{M=-J}^{J} \sum_{M'=-J}^{J} \hat{f}_{MM'}^J D_{MM'}^J(\alpha, \beta, \gamma). \tag{5.22}$$

Here, the orthonormal bases $D_{MM'}^J$ are the Wigner-D functions, and $\hat{f}$ are the coefficients of the SOFT.

Comparing (5.22) with (5.21), we see that the $SO(3)$ Fourier coefficients of C are directly related to the spherical Fourier coefficients $\hat{f}$ and $\hat{h}$. With this, an inverse SOFT yields values of C sampled on a $2B \times 2B \times 2B$ grid. The grid is discretized according to $\alpha_j = \gamma_j = \frac{2\pi j}{2B}$ and $\beta_j = \frac{\pi(2j+1)}{4B}$, showing the relationship of B to the rotation estimate. The desired Euler angles are now known, given by the indices which indicate the maximimum value of the correlation in the grid.

The spherical harmonics module is based on the SOFT routines provided by Kostelec and Rockmore [115]. We use a bandwidth of $B = 64$ in our experimental implementation.

5.7.2 Experimental setup

We compare two variants of a feature-based ego-motion estimation algorithms, two ICP variants, and the spherical harmonics approach to evaluate their performance. For the first feature-based approach, we use RGBDSLAM [45], a popular open-source implementation of 3D SLAM built using the ROS framework. Since the algorithm is a simultaneous localization and mapping algorithm and includes loop closing, we run a version of the algorithm to isolate just the ego-motion estimation to make the comparison more fair. The other algorithm is a custom in-house variation of the same approach, based on the outline in Algorithm 1, described below. To highlight the benefits of the spherical harmonics approach compared to plain ICP, we utilize

two standard algorithms: the traditional point-to-point ICP of Besl and McKay [13], as well as the probabilistic point-to-plane generalized ICP (GICP) of Segal et al. [186].

The custom feature-based algorithm extracts features from the color images of the RGB-D sensor. Matching features are found between two frames using an L_2 distance metric on the feature descriptors. We use SURF [8] keypoints and descriptors from the OpenCV library [18]. These extracted keypoints are projected into a 3D point cloud, and a RANSAC algorithm is used to estimate the best set of inliers corresponding to the initial three point estimated transformation. We then apply singular value decomposition (SVD) to solve the least-squares estimation of transformation parameters to the inlier set, yielding the egomotion estimate.

5.7.3 Data sets

The first data set collected was designed to evaluate the performance of the spherical harmonics algorithm independent of the ICP post-processing step (to account for translation). To collect the data, a Kinect sensor was mounted to a pan/tilt unit, which controlled just the rotation of the sensor. These experiments were carried out in our indoor testing environment. The testbed is surrounded by a Vicon motion capture system, giving time-synchronized ground truth data. The unit was controlled to rotate at a continuous speed about the sensor frame y-axis.

To analyze more challenging data, we chose to test the algorithms using two of the benchmark data sequences proposed in [196]. These two data sets, which are sampled at 10 Hz, are also time synchronized with motion capture measurements for ground truth. The data were taken in a real (cluttered) office environment, and the sensor is moved by hand throughout the office. The resulting sensor motion is at times more chaotic than it would be on a typical indoor robot, and these sets were expected to prove particularly challenging to image feature matching algorithms. Figure 5.2 shows benign and challenging example images from one data set.

The algorithms were evaluated on a 2.5 GHz laptop.

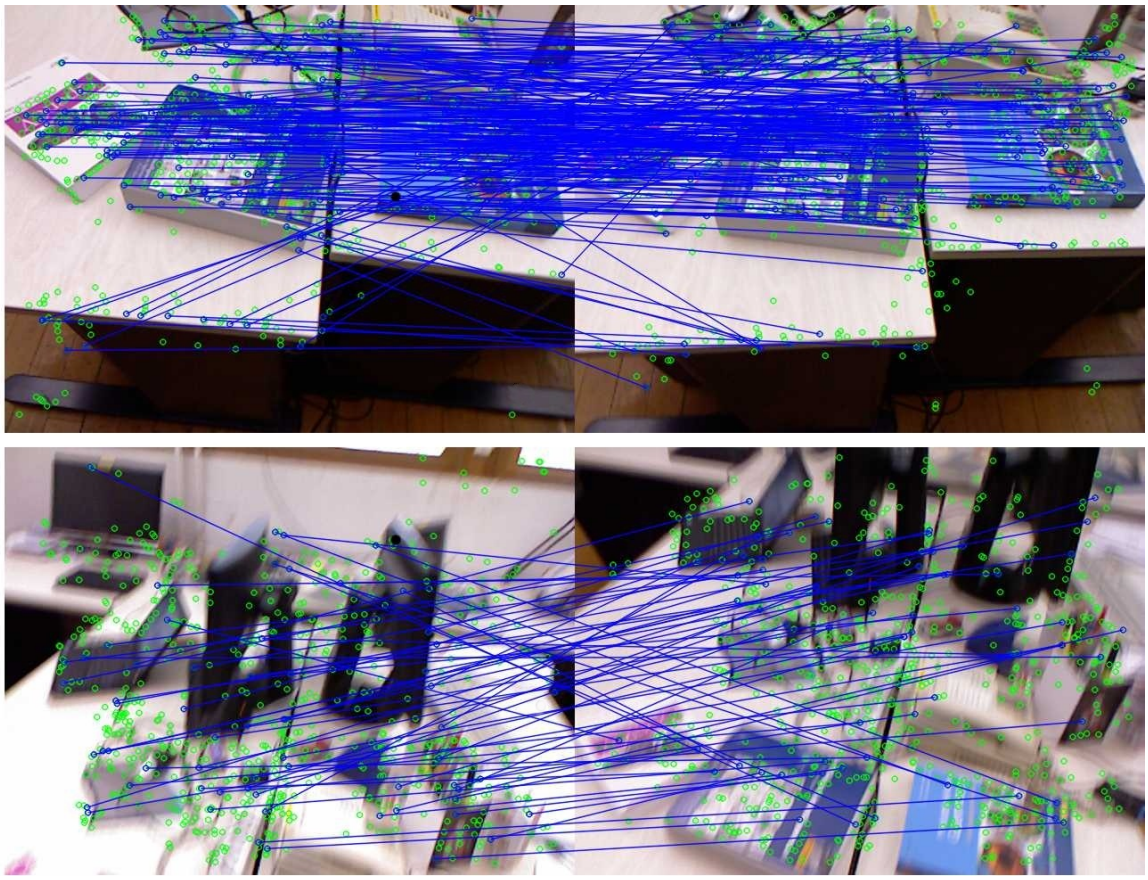


Figure 5.2 From the external benchmark data set, "360_10hz.bag", the difference in reliable feature matching between benign (top) and aggressive (bottom) sensor motion.

5.7.4 Results

In this section, we examine the performance of the spherical harmonics-based ego-motion estimation with three other VO variants and ground truth. We use translation and rotation error and runtime metrics for comparison. The following results analyze a measure of accumulated pose accuracy over time, a measure of the accuracy between frames of data, and the runtimes for each algorithm. For the first data set, we compare the value of the total rotation about the sensor y-axis, as well as the corresponding frame to frame angular error. For the next two data sets, we compare the total translation along an axis, along with the frame-to-frame pose error. The total translation is signed error along a single axis, while the frame-to-frame pose error is the magnitude of the vector between the estimated and ground truth pose changes. Finally, we analyze the performance of the algorithms on additional data

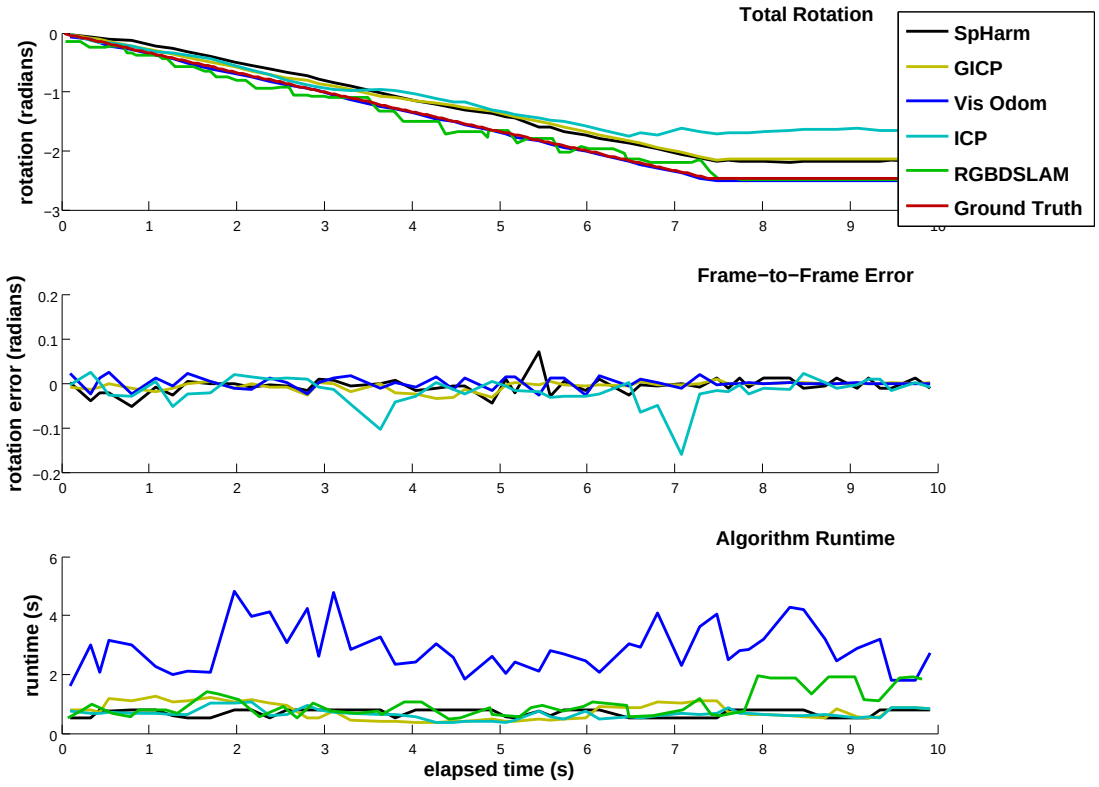


Figure 5.3 In-house data set testing results. While the in-house visual odometry takes longer on average to run, its results are more accurate than any algorithm. ICP fails to converge, resulting in large frame-to-frame errors, while RGBDSLAM uses correction steps to reach the final total rotation estimate.

sets and summarize the results.

The results of the first data set isolating rotations are shown in Fig. 5.3. The differences between a SLAM-based algorithm and pure egomotion algorithms are apparent in the total rotation plot. The discrete steps in the RGBDSLAM plot indicate a correction has taken place. While non-correction steps from SLAM yield incorrect results, the correction step often achieves lower global error than the other algorithms. The spherical harmonics approach runs fastest on average, while the custom feature-based VO yields the lowest frame-to-frame error. Note that the point-to-point ICP algorithm fails to converge in at least one frame, illustrating a weakness of ICP. Since the GICP algorithm outperformed point-to-point ICP in all of our experiments, we drop the standard ICP and only analyze GICP performance in the remaining results.

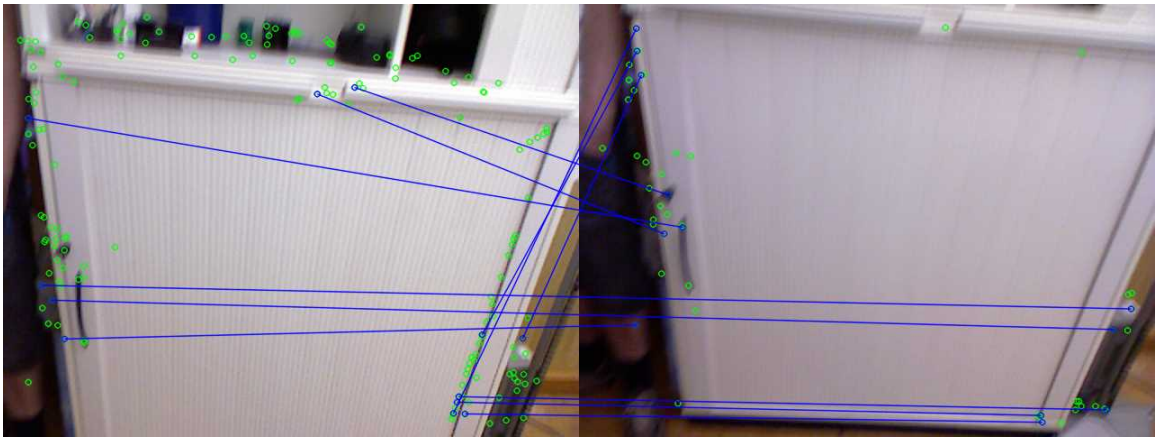


Figure 5.4 From the external benchmark data set, "room_10hz.bag", consecutive frames with a very sparse feature set will cause visual odometry estimation to fail.

The next data set (from Freiburg [196]) contains data yielding the weakest image features, illustrated in Fig. 5.4, of all the data sets analyzed, implying the performance of the feature-based algorithms derived from appearance should be reduced. The custom VO algorithm failed in this case, and to overcome this, GICP was added to the VO process, as well as a relaxation of the RANSAC constraints. The relaxation led to improved runtimes but at the cost of reduced overall accuracy. Likewise, there are frames in which the Spherical Harmonic process yields a completely incorrect rotation. The reasons for this were discussed in [136], and are related to the fact that the algorithm computes a single correlation based on noisy, weighted data in a discretized grid. Our process currently uses multiple rotation hypotheses similar to [136], and further work should investigate alternate methods of detecting and handling incorrect results.

The results of the third data set, `room_10hz.bag`, are shown in Fig. 5.5, with total translation measured along the sensor frame z-axis. The GICP process runs faster than the others, but has the highest frame-to-frame error. The Spherical Harmonics process has the highest frame-to-frame accuracy, and its total accuracy is approximately equal to that of the RGBDSLAM algorithm. The VO with GICP process and the GICP process plots are quite similar, which may indicate that errors in GICP are affecting the accuracy of the VO with GICP process.

We see similar results in the final data set, `360_10hz.bag`, in which the sensor

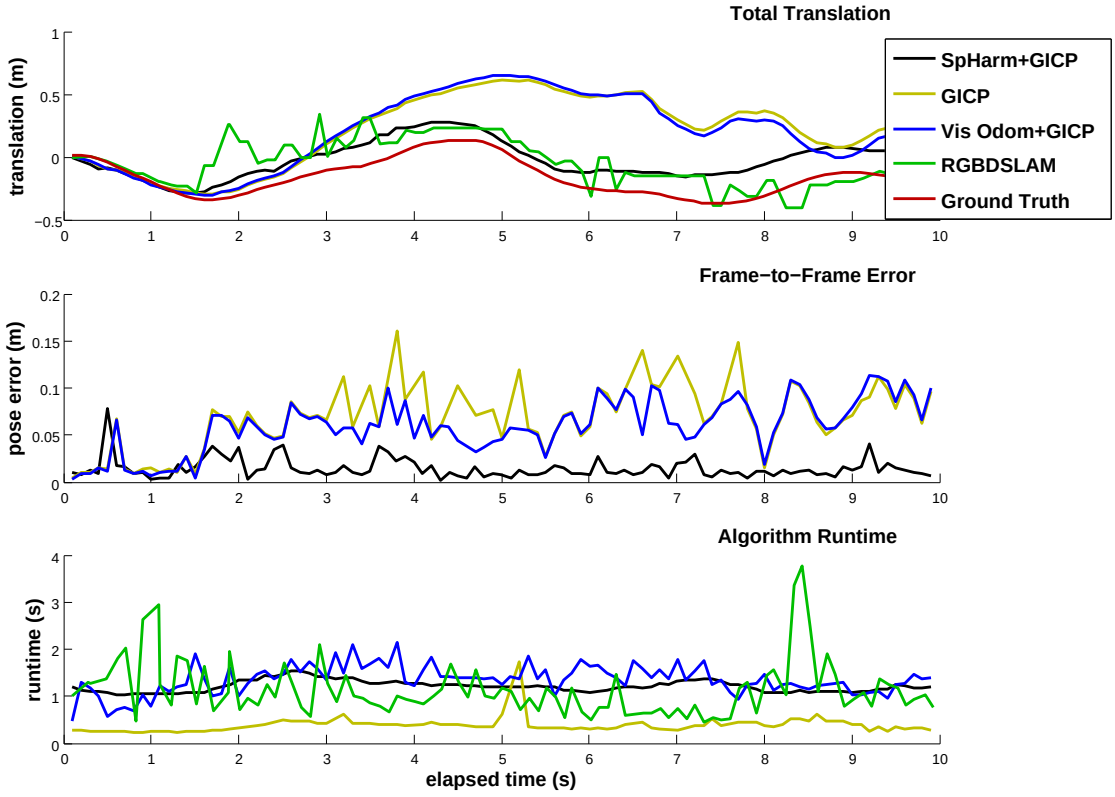


Figure 5.5 Results comparing the performance of the algorithms on the 10Hz room benchmark data set. Spherical Harmonics+GICP has the highest frame-to-frame accuracy, and comparable total translation value as RGBDSLAM.

is being rotated around a room and simultaneously tilted between ceiling and floor. The sinusoidal motion can be seen in the results, shown in Fig. 5.6. The benefits of the correction steps of RGBDSLAM are evident in the plot, and contrast with the accumulating errors of the ego-motion algorithms.

To determine the effectiveness of the algorithms when presented with wider baseline camera poses, the benchmark data sets were downsampled and re-analyzed. Figure 5.7 shows the results of sampling every other frame for the second and third data sets. As with previous results, Spherical Harmonics achieves higher accuracy on one data set, while visual odometry is more accurate on the other. Figure 5.7 also shows the results of the second data set, sampled every third frame. Early in the data set, RGBDSLAM fails to match new features to old features. Although frame-to-frame

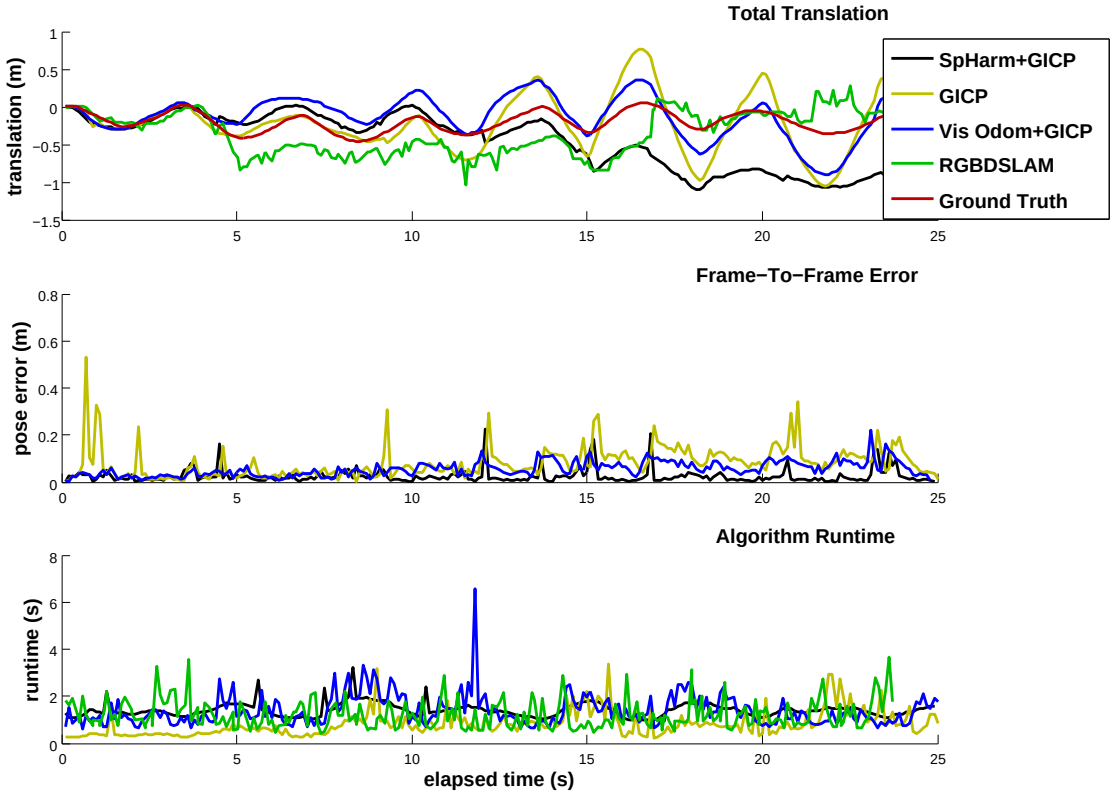


Figure 5.6 Results comparing the performance of the algorithms on the 360°10Hz benchmark data set. Spherical Harmonics+GICP again has the highest frame-to-frame accuracy. RGBDSLAM is initally inaccurate, but eventually corrects to an accurate total translation.

matches can be achieved in future frames, the features cannot be globally matched to features from the initial frames, and as a result, RGBDSLAM does not recover after the initial failure. The custom VO defaults to GICP as discussed, and can recover in the presence of a failed attempt at feature matching.

To generalize the results, we ran the algorithms over eight additional data sets, all obtained from [196]. The results are summarized in Table 5.1 and Table 5.2. As expected, the combination of Spherical Harmonics and GICP is more accurate than either algorithm alone, and the Spherical Harmonics algorithm alone achieves a very consistent, low runtime. Finally, Fig. 5.8 shows example point cloud reconstructions of the first three data sets using each algorithm.

Table 5.1 Average Frame-to-Frame Rotational Error (radians)

	SpHarm	SpHarm GICP	GICP	VO GICP
mri_downstairs	0.0177	**0.0100**	0.0115	0.0116
freiburg1_room	0.0679	**0.0227**	0.0945	0.0880
freiburg1_360	0.0444	**0.0222**	0.1208	0.1223
freiburg1_floor	0.0710	**0.0136**	0.0150	0.0154
freiburg1_xyz	0.0369	0.0105	**0.0104**	**0.0104**
freiburg1_rpy	0.0635	**0.0237**	0.0323	0.0268
freiburg1_desk2	0.0640	**0.0261**	0.0627	0.0539
freiburg2_xyz	0.0588	0.0090	**0.0073**	**0.0073**
freiburg2_rpy	0.0518	0.0132	0.0118	**0.0104**
freiburg2_desk	0.0592	**0.0137**	0.0151	0.0142
large_no_loop	0.0477	**0.0205**	0.0252	0.0209

Table 5.2 Average Runtime (seconds)

	SpHarm	SpHarm GICP	GICP	VO GICP	RGBD SLAM
mri_downstairs	**0.3497**	1.0451	0.6631	2.5931	1.1589
freiburg1_room	0.3423	0.7149	**0.2987**	1.5311	0.8076
freiburg1_360	**0.3438**	0.8661	0.4235	1.5384	0.9441
freiburg1_floor	0.3416	0.6764	**0.2388**	1.2115	1.1707
freiburg1_xyz	0.3410	0.7147	**0.2931**	1.3636	0.8787
freiburg1_rpy	**0.3423**	0.7735	0.3683	1.3719	1.1267
freiburg1_desk2	0.3440	0.7060	**0.2715**	1.1700	0.9940
freiburg2_xyz	0.3421	0.7182	**0.2912**	1.3888	1.3020
freiburg2_rpy	**0.3433**	0.9186	0.5154	1.6989	1.0834
freiburg2_desk	**0.3444**	0.8659	0.4693	1.7980	1.2435
large_no_loop	**0.3506**	1.3819	0.8920	3.0711	1.0411

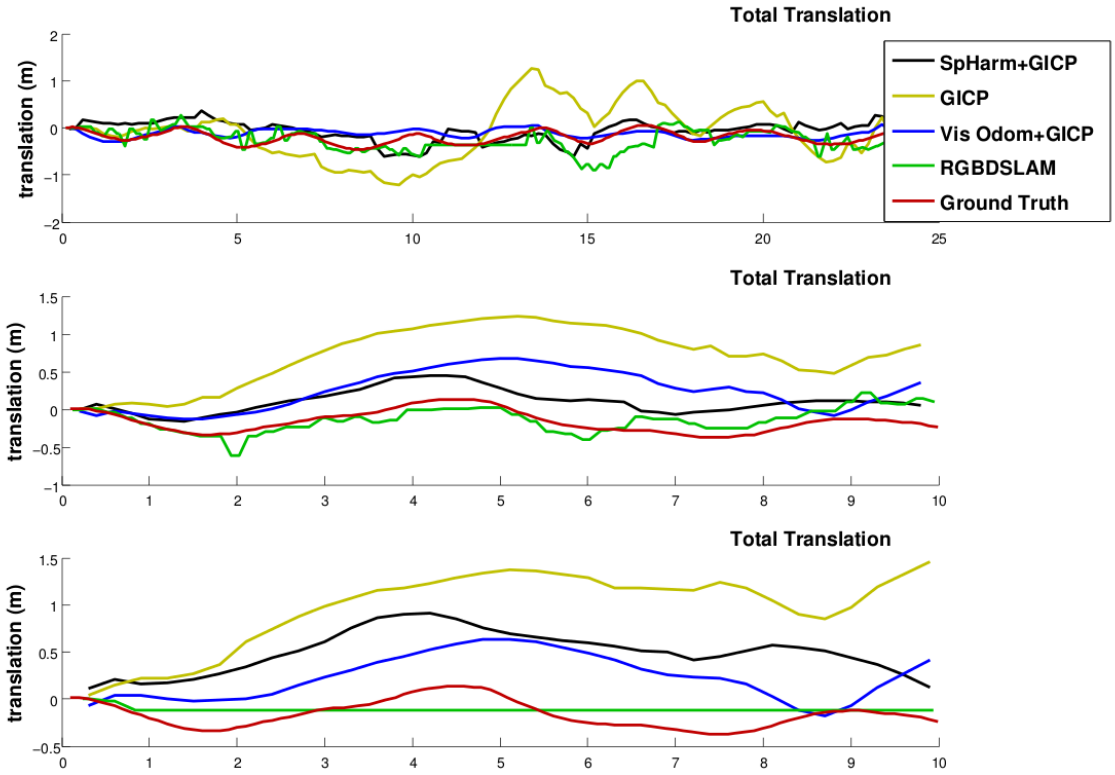


Figure 5.7 Results from the second and third data sets, after downsampling. At top, the third data set is sampled every other frame. Below, the second data set is sampled every other frame (middle) and third frame (bottom).

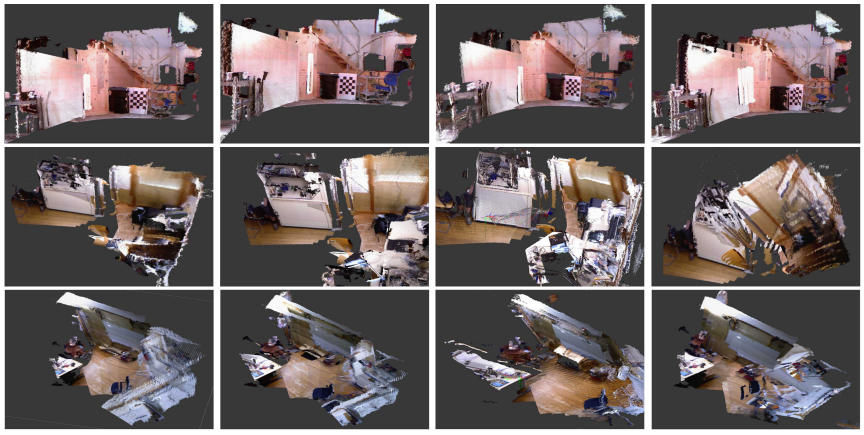


Figure 5.8 Reconstructed point clouds. Each column is the result of reconstructing data from the three data sets analyzed in detail. From left to right, the algorithms used for reconstruction are: Spherical Harmonics+GICP, Visual Odometry+GICP, RGBDSLAM and GICP.

5.8 Conclusion

We conclude this chapter with a summary of the various approaches, and a direction for the future.

5.8.1 On joint optimization

The joint DVO-ICP optimization approach improved on each of the individual methods, and allowed the system to generate the accurate models shown in section 6.3. However, it is certainly not the final answer. As I mentioned previously, the fact that DVO is based on the photometric constancy assumption means that it is sensitive to properties of the camera as well as the scene illumination, which can cause the camera to adjust the exposure and white balance between frames enough to violate the assumption. This was the largest source of failure for the algorithm, especially in scenes with low lighting or larger variations in illumination intensity. In recent visual odometry papers using the direct approach, greater emphasis *has* been placed on characterizing the camera lighting response and correcting for or controlling exposure variations between frames, leading to greater accuracy in challenging conditions [12], [50], [234]. This trend should continue, although it makes the process more complex, and should bring additional methods into the fold as well; i.e., simple to compute indirect methods that may be more robust to lighting as well as the use of intermediate or semantic features like contours and objects.

Another consideration, as mentioned in the discussion of Fig. 6.4, is that loop closing processes for mapping need a secondary alignment algorithm to handle the general case of matching two scenes from not only arbitrary viewpoints, but unknown poses. This would likely *require* an indirect approach, or something like spherical harmonics, to provide an approximate alignment at the minimum, followed up by the joint optimization.

5.8.2 On spherical harmonics

Ego-motion estimation using spherical harmonics has the potential to improve on the performance of feature-based algorithms, especially in challenging environments where there is geometry present but few features detected. This is similar to the benefits of combining ICP with DVO, but where spherical harmonics can find the rotation even when the near-frame projective assumption is violated. While spherical harmonics also tended to be more efficient than some of the algorithms we compared in section 5.7.4, it is not as efficient as the joint DVO-ICP algorithm we optimized on the GPU. However, work such as that done by Schaeffer [183] to implement fast spherical harmonics transforms on GPU could provide a route to integrating the approach with the joint algorithm.

5.8.3 Looking to the future

I do not think it is contentious to assert that robots need highly accurate ego-motion estimation, if only to integrate local views of their environment into a coherent model (this is discussed in the next chapter). While the algorithms discussed in this chapter present viable options, they each have benefits and drawbacks, and not one provides a robust *enough* solution in and of itself. Instead, the way forward will be to continue to integrate: specifically, we must find ways to efficiently make use of *all* the information available to us: environment appearance and structure as well as proprioception (e.g., IMUs and wheel or limb actuation). We cannot tolerate failure in this task; the system must be robust to any condition, and operate to at least human-level standards in challenging environments. I propose the following requirements for ego-motion estimation: in **any** 4×4 meter environment, the system should maintain localization accuracy of less than 1 cm and 1 degree error given at least 1 landmark in view. These constraints should make it possible to move and manipulate objects within reasonable error bounds, and prevent catastrophic motion estimation and mapping failures such as those from GICP in Fig. 5.8.

Chapter 6

High-resolution Local Mapping

6.1 Introduction

For a robot, mapping is the process by which an agent generates a spatial[1] model of its world, based on the information it gains from a collection of sensor measurements and a method of representation chosen to best support the goal of the model. For example, we know that humans generate representations of the world in the form of cognitive maps [30], [102], [220], [227] (although what specificially constitutes those maps and how they are generated is still an area of research). In addition, humans have been generating maps for thousands of years as external artifacts that document what we know about the planet and the places in which we live. In everyday experience, we use map artifacts posted in shopping malls, subways, and rail cars to determine how we will navigate within the relevant environment. They provide a representation that we can use to suitably plan our behavior, such as route following.

Mapping is a key component for understanding and *operating within* the world since it enables navigation through planning. Any embodied intelligent agent needs to be able to find its way in the environment, and without some model of the surroundings, it would be very difficult. Imagine not having a model (i.e., map) of your own living space; how would you be able to function effectively? Every time you needed

[1] I emphasize *spatial* here, since there are other ways a robot may model the world; e.g., causal relations between actions and effects, physical properties of motion like gravity and inertia, social behaviors of humans, and many others.

a bowl to eat or needed to use the restroom, you would have to search randomly. It is actually hard to imagine this scenario, since this kind of spatial and semantic modeling is such an inherent human capability. We argue that building these models is also a requirement for any embodied system that must carry out arbitrary tasks over long periods of time. Experience should allow the system to build up effective spatial models of the world so it can use them to plan to navigate as efficiently as possible with respect to the current goals of the system.

Mapping, or environment modeling[2], has another more fundamental purpose as well: providing a basis for understanding the spatial structure of the world. We conceive of these models in the following terms based on this thought experiment: imagine you are an intelligent autonomous system (without access to human intelligence) and you possess a set of sensors that provide a snapshot of information about the world[3]. Let us interpret a single snapshot *as* a model of the world. This model may be useful in itself, depending on a variety of factors. However, its utility may be directly proportional to how much reasoning and planning you can do with it. If this snapshot happens to be an organized set of colored pixels (i.e., an image), then it may be very hard to do *anything* with it, without some significant experience and learning algorithms that allow you to interepret the image in a way that enables you to achieve your current goal (e.g., exit the room). If you are looking towards a doorway already, and you have an algorithm that can recognize doorways, then perhaps you can generate a motion command that would move you towards the doorway. However, did you recognize the heavy object (say, a desk) that happened to be in the way? A more informative sensing modality (such as a depth camera or laser range finder) may have given different information that is more directly usable (i.e., obstacles to avoid, whether recognized or not), and therefore you may have been more successful with this single snapshot.

In the same vein, consider the ability to merge snapshot after snapshot (say, at

[2]Throughout this chapter, I shall use these terms interchangeably, as they arguably mean the same thing. However, we tend to think of environment modeling as a special kind of mapping that focuses on more accurate representations of the (local) environment.

[3]Please allow me to use this discretized metaphor, as it is easy to talk about and relate to the sensors and systems we work with.

a rate of 15 Hz) into a single coherent representation that respects your ego-motion and the actual structure of the world. It would then be possible to "look around" and build up a larger representation that is more useful than any individual part on its own. It is easy to imagine the utility of this kind of representation at scale, providing enough information to plan a path to a goal through an entire room, building, or city. It would simply not be possible to do so without this model. In contrast, if we imagine a robotic insect that cannot build these representational models and can only make choices based on the incoming snapshots, one at a time, then we can see how difficult it might be to reason about future behavior. Yet, reasoning about future behavior in order to achieve goals is one part of intelligence.

6.1.1 Kinds of maps

There are several different kinds of maps (and many different kinds of algorithms) that have been used for motion planning. In this section, we give a brief overview of the map types useful for understanding the place our mapping algorithm fits in the overall picture.

How many dimensions?

Maps can be represented in either 2 or 3 dimensions. In the 2D case, occupancy grids are often the default. The environment is assumed to be flat (or at least, it is acceptable for the representation of the environment to be perceived as flat), and is subdivided into equally spaced regions (also called cells) represented by one or more values stored at each x, y coordinate. In the typical case, the value stored at a location represents the likelihood of that cell being occupied. If a cell is occupied it contains some kind of object that would prevent the robot from moving through that cell. In this 2D planar world, motion planning usually occurs in 3 dimensions: (x, y, θ), representing the x, y translation relative to some origin, and the yaw *theta* representing the robot's heading.

The 2D representation is simple, easy to represent, and straightforward to nav-

igate; however, it does have some limitations. You might realize the fact that our world is, in fact, not 2D, but actually 3D. This means it is not possible to represent the height of obstacles and objects in the 2D occupancy grid, and it is not directly possible to represent situations such as the floors in a building or the hills in a field. We have even encountered environments that are mostly flat (another person would generally agree), but still cause problems with sensors such as 2D laser range finders because they are not *actually* flat: specifically, the imperfections of the surface caused the robot to pitch enough that the plane of the laser sensor intersected the ground meters in front of the robot, generating false obstacles in the occupancy grid [160]. Additionally, the accuracy of your model depends on the resolution of the grid. However, increasing the resolutions requires quadratically more memory and quadratically increasing time for many planning algorithms, as halving the resolution generates 4 times as many cells. One method that may be used to account for this involves representing the occupancy grid in a more efficient data structure, such as a quadtree [58].

In 3 dimensions, there are a variety of ways to represent a map. The key element, however, is that all three dimensions are present, and therefore, any environment can be modeled as accurately as the representation, sensors, and algorithm will allow. In addition, in 3D dimensions, navigation can be performed through all 6 degrees of freedom, $(x, y, z, \theta, \phi, \psi)$. In practice, whether on a ground or air robot, planning and motion are still relative to the local ground surface, which can be considered a 2D manifold embedded in 3D space.

What is modeled?

In the previous section, we went into some detail about a *particular* 2D map representation. There are others, and the representations are relevant to both the 2D and the 3D case. The 2D occupancy grid has a 3D analog: the 3D occupancy grid, usually stored in a multi-resolution spatial data structure such as an octree [140] (also, see section 7.2.1).

The primary dichotomy in representation form is between dense, grid- or point

cloud-based representations and sparse, landmark-based representations. The 2D and 3D occupancy grids are examples of dense grid representations. A point cloud-based representation can also be considered dense, but does not explicitly represent the "free space", although it can usually be computed based on the viewpoint of the agent. In addition, point cloud representations don't depend on a grid-based resolution, as they are data structures containing points, and therefore can be more efficient than an octree-based occupancy volume for the same information. Another dense representation, popularized originally in KinectFusion [99] but used extensively thereafter [38], [122], [128], [153], [214], [216], [217], is a TSDF volume [36]. Similar to an occupancy grid, this representation is particular amenable to merging multiple depth maps (R_k) interpreted as rays cast into a volume. Each depth reading represents a noisy measurement we assume can be *truncated* by some value μ, such that cells along the ray $\mathbf{r} < \lambda R_k(\mathbf{p}) - \mu$ are considered free space, and similarly cells along the ray $\mathbf{r} > \lambda R_k(\mathbf{p}) + \mu$ are considered occupied space. The cells along $\mathbf{r}$ are uncertain but are stored as a signed distance function where the positive values are outside the surface, the point at 0 represents the surface, and negative values are inside the surface. This allows an uncertain representation of the surface that can be updated by new readings, allowing the 0 crossing to change as appropriate based on weighted averaging of the cell values.

At the other end of the spectrum, landmark-based representations allow sparsity because they only store points or objects in the environment that have been detected, and then, only those points or objects that are likely to be robust to different viewpoints or lighting conditions. While these representations can still be useful for pose estimation and mapping, it is much harder to generate path plans, since the actual structure of the environment is not explicit.

6.1.2 High-resolution 3D map

We choose to generate a high-resolution 3D map, but this choice comes with benefits and compromises. A high-resolution cloud of surfels provides enough information for segmentation algorithms that may be involved in online object learning; the fact

that multiple frames often contribute to single surfels means the overall noise is reduced, generating better boundaries between objects and fewer holes in the model. High-resolution models provide more detail, meaning smaller objects can be resolved. Relationships between objects provide information about semantic associations and typical spatial organization of related objects.

On the other hand, high-resolution models make it difficult to manage the size of the objects in memory (whether CPU or GPU, although the latter is usually more constrained). Also, while sufficient information is present to support navigation, working with such a dense point cloud is difficult, and still requires processing into something simpler before it can be used effectively for path planning (e.g., generating a low-polygon mesh for collision detection).

In the rest of this chapter, we document our approach to dense, high-resolution 3D mapping using RGB-D sensors, which we call the surfel modeler. We chose this approach to support learning online object recognition tasks (i.e., by capturing partial object models) and to support learning about spatial relationships between objects and regions[4]. Our process relies directly on the incremental processing of dense depth maps provided by an RGB-D sensor, and also requires locally accurate ego-motion estimation.

6.2 Approach

It is fairly clear that mapping is inherently tied to ego-motion estimation. Consider the phrase "merge snapshot after snapshot into a single coherent representation that respects ego-motion and the actual structure of the world." The first part is "single coherent representation," and the second part describes the two constraints. The first constraint means that the way we integrate the information snapshot into the whole must respect the information from the system's motion... this is metric information that describes the sensor pose at individual time instants and how the pose changes

[4]While these topics are outside the scope of this dissertation, we present some ideas for future work in these areas at the end of the chapter.

from one instant to the next. This is critical information and makes the task of merging snapshots easier; if we have it, then we know how we have moved with respect to the previous snapshot, and can use the knowledge of the motion to better align the snapshots for mapping. The second constraint means that we want the whole to be a good representation of the world, and allow for reasoning algorithms that can directly translate into action within the world. This depends heavily on the ultimate goals for the representation; for example, if the only goal is to avoid obstacles and you are a ground robot (in one of many guises), you may not need a representation that models the full 3D structure of the world, or the height of objects, since you just need to see where there is space to move on the ground; in other words, a 2D occupancy grid would suffice. On the other hand, if you are an aerial robot, or need to understand and manipulate human-scale objects (like the DARPA Robotics Challenge [39]), then a 3D model of the world is required.

SLAM is the typical algorithmic technique used to generate maps when exploring unknown environments. It relies explicitly on receiving ego-motion updates from an odometry component (likely visual odometry, as discussed in chapter 5) and then uses this information along with the information in the sensor snapshots to incrementally build up a model of the world. A key aspect of SLAM is the iterative nature of the task, but also the ability to correct for ego-motion errors by recognizing previously visited places through the process of loop closing. This allows one to create a connection or constraint between places, and facilitates the correction of drift in the model using a process of graph optimization.

6.2.1 Ego-motion and Localization

Ego-motion and localization are related but separate concepts. Ego-motion simply refers to a description (or computation in the case of ego-motion estimation) of how a sensor or agent is moving, while localization refers to the task of determining a pose given a map and sensor data. Ego-motion, in fact, can be used to help determine and maintain the localization within the map, since knowing the local motion can be useful to compare the sensed observations with the expected change in the observations of

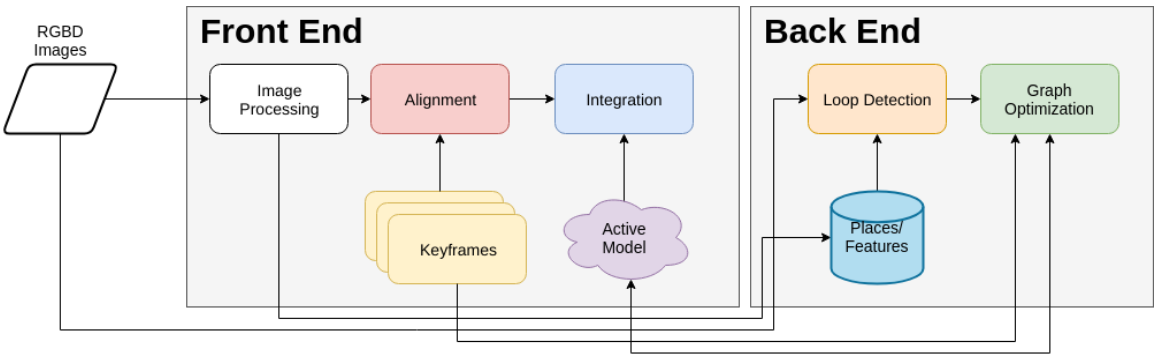


Figure 6.1 Our SLAM system diagram. While some SLAM systems can accept different kinds of sensor input, our algorithm focuses on RGB-D frames, that is, a pair of RGB and depth images.

the map features based on the estimated location.

In the previous chapter, we described an ego-motion algorithm that computed the incremental pose between two sensor frames. In this chapter, we describe how we can use this technique in concert with building maps to perform both ego-motion estimation *and* localization within the map we are building (implementing a version of a SLAM algorithm).

6.2.2 SLAM architecture

Our approach follows the typical structure of modern SLAM algorithms: the *front end* component computes the ego-motion as frames enter the system, and integrates new depth maps for each frame into the currently active portion of the map. The *back end* does loop detection, and optimizes the pose graph upon loop closures.

Figure 6.1 illustrates the overall architecture of our SLAM system. The following subsections will describe the function of each block in the diagram.

Front end

The front end contains many components we described in Chapter 5. RGB-D frames are submitted to the algorithm as they are received from the sensor, and first undergo image processing (see section 5.6.2) to prepare them for the alignment step (see section 5.6). As mentioned in section 5.6.1, incoming frames are aligned not necessarily

with the most recent frame, but with keyframes instead. This tends to improve the accuracy of the ego-motion estimation, and directly contributes to the maintenance of the active model using integration, described in a following section. First, however, we describe our modeling element, the surface element.

Surface Elements (Surfels)

A surface element s (shortened to *surfel* like pixel for picture element) is a generalization of a (usually) small region of a surface represented as the set $s = \{\mathbf{p}, \mathbf{n_p}, r, \mathbf{c}, \mathbf{v}_h, \kappa, C\}$, where $\mathbf{p}$ is the point indicating the centroid of the region, $\mathbf{n_p}$ is the normal of the surface at the point $\mathbf{p}$, r is the radius of the region disk[5], $\mathbf{c}$ is the color of the surface, $\mathbf{v}_h$ is the histogram of viewpoints the surfel has been observed from, κ is the curvature of the surface at $\mathbf{p}$, and C is the confidence in the existence of the surfel.

Why are surfels interesting for high resolution 3D modeling? Surfels are a surface sample (and as such, have area), which is semantically more pleasing than points, and better represents the imaging process from the camera. In addition, surfels allow for generating surfaces at varying resolutions, depending on how far away the camera is to the observed surface. Surfel clouds, like point clouds, do not require volumetric storage representing all the empty space, as opposed to popular dense representations such as TSDF volumes (see section 6.1.1).

On the other hand, surfels do take up more space, and like point clouds, do *not* have a direct way to represent neighborhood connectivity when represented in a typical unorganized cloud.

Integration

Integration is the process of using the ego-motion estimate to adapt the incoming point clouds (derived from the depth image) into the existing active environment model. We attempt to reduce error accumulation in the model by continuously integrating incoming frames into a single model composed of surfels, and subsequently use that model for the ICP-based motion estimation in the following frames through

[5]The simplest and most common region shape.

the extraction of a model-based keyframe. The model is created and updated on the GPU, to maximize throughput when visiting individual surfels.

The algorithm is initialized with the first frame from the camera set as the initial model. Following query frames are aligned using the method described in the previous section, where the GPU-based join DVO/ICP algorithm aligns every visible point-normal pair $(\mathbf{p}, \mathbf{n_p})$ derived from the surfels in the model to the query frame using the GPU's bilinear interpolation texture hardware. Once an estimate is generated, the query frame is integrated into the model.

Surfel integration Integrating the query frame into the model is performed in two steps: update and addition. Update is performed first, followed by the addition of points in the query frame that were not processed during the update procedure. This surfel integration procedure was adapted from the work of Weise et al. [211].

Update: Given an estimated transform $\mathbf{T}$ that maps the model frame to the query frame, we can iterate over every visible surfel $s_i \in \mathcal{M}$ and project it onto the query image plane, $s_i' = \mathbf{T}s_i$. Since the camera is moving and previous observations may be noisy, a query point may be in front of, behind, or unobserved with respect to a projected surfel. Part of the decision to update is made based on the distance between the model surfel and the query point, as well as the difference in their normals. The following conditions determine the outcome. Let $d = z(s_i') - z(p)$ and $\alpha = \nu s_i' \cdot \nu p$. We also have the parameters $D_{\max}$ representing the maximum surface offset allowed, and A as the minimum dot product between normals.

$|d| < D_{\mathbf{max}}$ The query point is within the maximum offset, and therefore the query point will be used to update the surfel values.

$d > D_{\mathbf{max}}$ The query point is in front of the model surface. This can be due to an extreme outlier, or the motion of the camera bringing another surface into view that is truly obscuring the view of the existing surfel. In this case, we do not update the surfel, and instead mark this query point to be added independently to the model in the next step.

$d < -D_{\mathbf{max}}$ The query point is behind the model surface. Since we are unsure whether the query point is an outlier or the model point is an outlier, we let our confidence in the surfel determine which action to take. If the surfel has been seen and updated multiple times (i.e. in other frames it has been seen at roughly the same location), then we assume the query point is an outlier and do not update the surfel. On the other hand, if the surfel has been viewed less than the minimum number of times (currently set to 3), we replace the existing surfel by generating a new surfel based on the query point.

Addition: For every marked query point, the second step simply adds a surfel derived from the point to the model. The point, normal and curvature are passed through unchanged (the latter two estimated by the PCL routine). The viewpoint is stored as the dot product of the normalized eye vector and point normal. The radius is estimated with the following equation:

$$\mathbf{radius}(d, f, z) = \frac{1}{\sqrt{2}} \frac{d/f}{z}, \tag{6.1}$$

where d is the depth of the point, f is the focal length of the sensor, and $z = \mathbf{n} \cdot \mathbf{e}$, so that the radius is larger the farther the viewpoint is from the normal direction.

Model Maintenance The GPU memory size is often more constrained than the CPU size, and over time the surfel model can grow larger than can be held in GPU RAM. This necessitates a swapping procedure, where a portion of the model is swapped out to be held in the CPU RAM. As part of the model generation, we maintain a list of keyframe locations (determined by the magnitude of the camera motion) and associate the keyframe locations with the surfels seen at each keyframe. We currently hold a fixed number of keyframes in memory, K_{mem}. At every keyframe trigger, the surfels that haven't been seen since keyframe $K - K_{\text{mem}}$ are swapped out to a surfel cloud that resides in the host (CPU) memory.

Back end

For the back end, we made use of the real-time appearance-based mapping approach for detecting loop closures of Labbe et al. [120]. They utilize hierarchical memory for storing place descriptors based on a bag of words approach for describing places. Features (i.e. words) are dynamically added to the vocabulary when they are sufficiently far enough away from existing words.

When the loop detection module discovers a loop, a new constraint is added to the graph. This new constraint triggers a graph optimization over the existing graph to bring the keyframes into alignment, and deforms both the camera poses and select control points embedded in the surfel cloud, based on the approach from Weise et al. [211]. We use the graph optimization framework GTSAM [105] in this work, as it comes with many features amenable to implementing SLAM optimizations.

6.3 Results

In this section, we highlight some of the typical qualitative results from running the surfel modeler on hand captured datasets. During development of this approach, we were less interested in how accurate frame-frame transforms were relative to an external source, and more interested in how well the system captured the details and structure in the environment. The samples in this section highlight some of the good and bad results achieved with the modeler.

The loop-closed living room model shown in Fig. 6.2 provides a good illustration of the detail achievable with the surfel cloud methodology. Note many of the pictures with clearly definable faces, along with accurate details on the fireplace and the couch. The loop is started and ended on the basket of blankets in the lower left corner.

This example also shows how errors in the photometric warping caused by changing exposure can affect the alignment, even when the ICP might provide good constraints. In this case, a slight misalignment can be seen at the TV in the upper center of the image, and is due to the camera automatically adjusting exposure to handle the reflection on the screen.

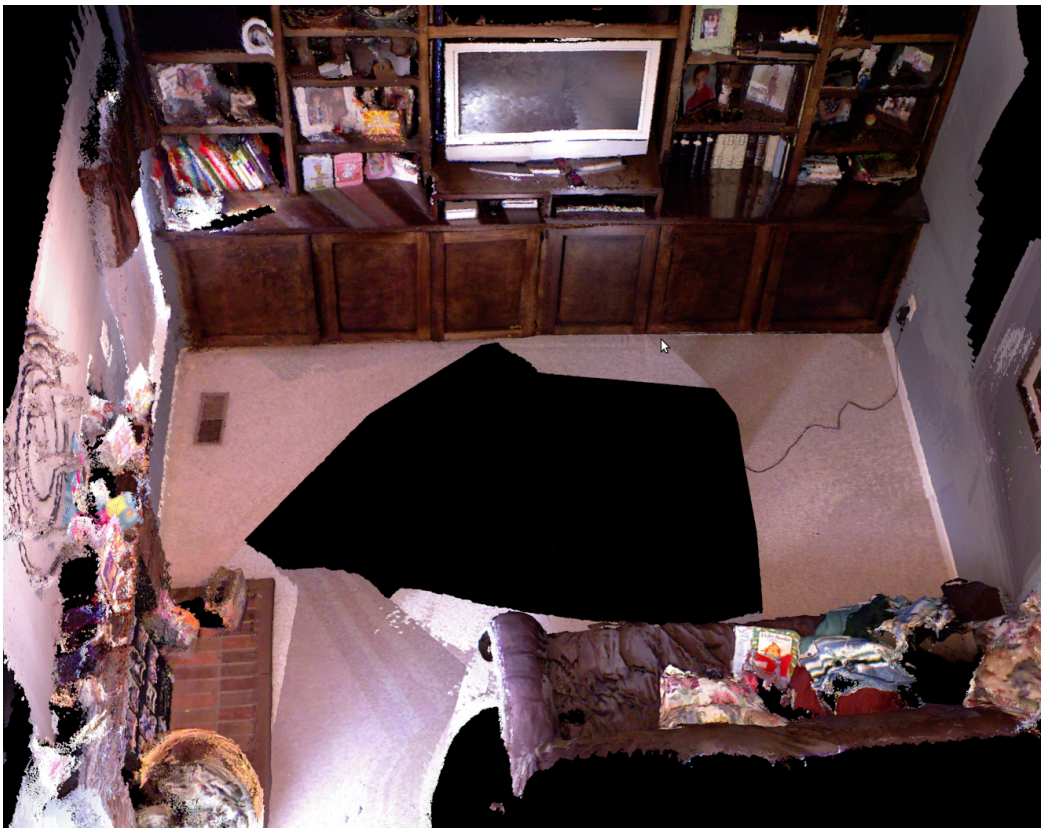


Figure 6.2 An example model of a living room captured using the algorithms described in this section.

In Fig. 6.3, we see an example of a model generated without global loop closure. Here, while individual regions align well, it is clear the ego-motion estimation accumulated drift, since the hallway walls are not parallel to the right office wall, as they are in real life. The recording began in the bottom right corner on the office chair and ended looking into the left office from the doorway. While the RGB-D sensors typically produce ranges between 5-7 meters, we discard any readings beyond 3 meters, since the data are often too noisy to be useful.

Figure 6.4 shows an almost closed loop of the office seen at the bottom left of the previous figure. Loop closure was often unstable when using the frame-frame alignment module; we suspect the primary reason for misalignment is that the two main assumptions are broken at a loop closure with even moderate drift. Specifically, (a) the lighting and exposure may not be the same, causing the photometric warping optimization to give incorrect results, and (b) the projective data association yields

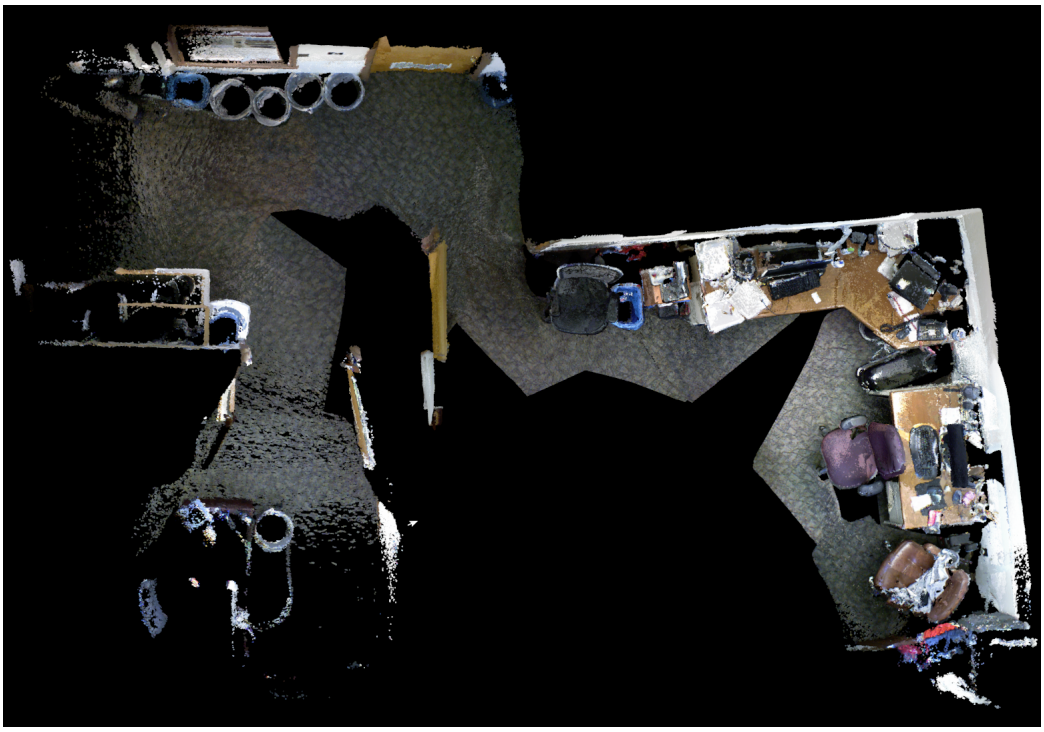


Figure 6.3 Sections of two offices and hallways. Notice the slightly bent hall; small amounts of error build up over time, requiring the application of a loop closing algorithm.

matches that may be too far apart and therefore also yield incorrect results, causing the alignment operation to fail. The algorithm could likely be improved by first substituting another frame-frame correspondence technique to bring the matching frames into closer alignment, and then running the joint dense alignment module to prepare for surfel integration.

The next model in Fig. 6.5 shows a relatively nice model of my children's playroom, but with an unsuccessful loop closure, which can be seen in the bottom center around the green play shopping cart with the maroon bag. The wall behind the cart is doubled, since when the loop was successfully detected, the drift was too large (on the order of 10 cm) for the alignment algorithm to close, as mentioned previously.

The surfel modeler also did a fairly successful (if slightly noisy) job of generating a nicely closed model of an object from continuous in hand scanning while circling the object, with no additional modifications. Figure 6.6 shows a cleaning product container with very little geometric variation from multiple viewpoints, illustrating

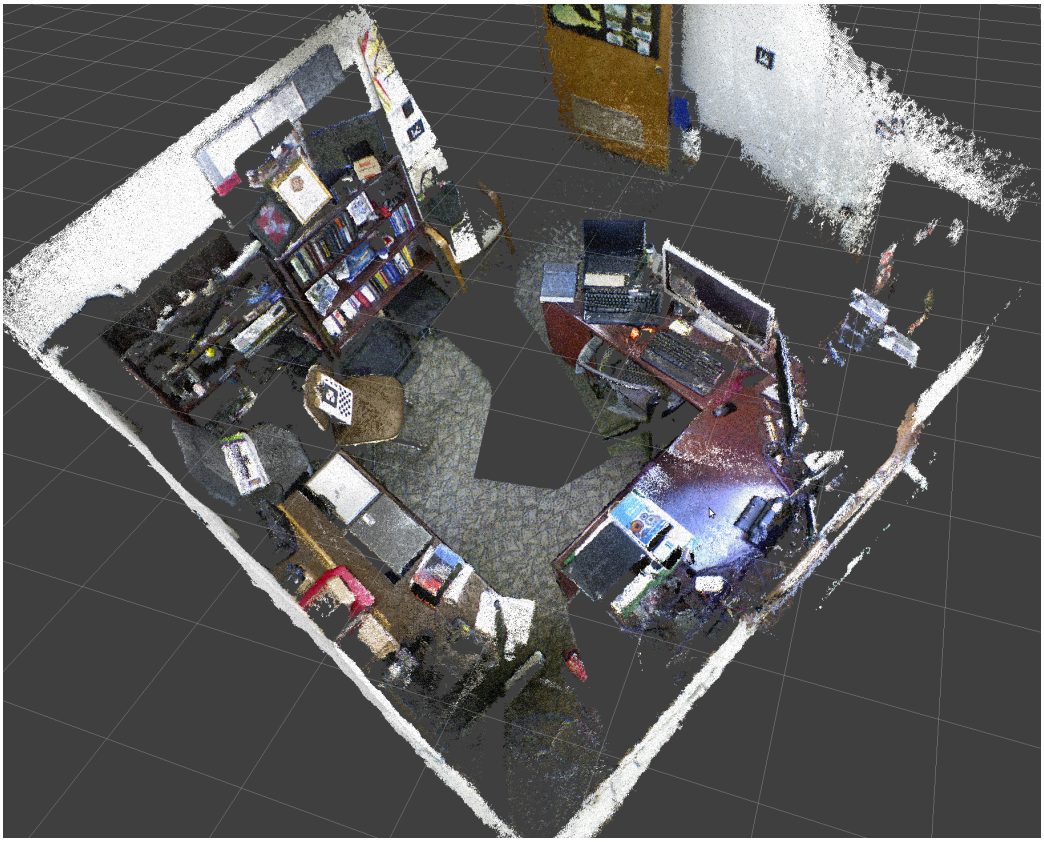


Figure 6.4 Another office model capture before the loop closure.

the detail maintained after many frame alignments and surfel integrations. The model is viewed within RViz, the standard ROS visualization tool, and uses square points as more efficient stand-ins for the circular surfel visualization.

6.4 Conclusion and Lessons Learned

The surfel modeler is an instantiation of the jointly optimized DVO/ICP ego-motion estimation algorithm with dense, high-resolution 3D surfel mapping. In the previous section, we showed several examples of model maps demonstrating high reproduction accuracy, but all is not consistently positive in the realm of surfel modeling. For example, we did not show many of the failures in mapping due to incorrect alignment caused by violated assumptions in the optimization algorithm. Since this work was performed[6], many systems have come out that in some ways are similar, and in other

[6]In the 2012-2014 timeframe.

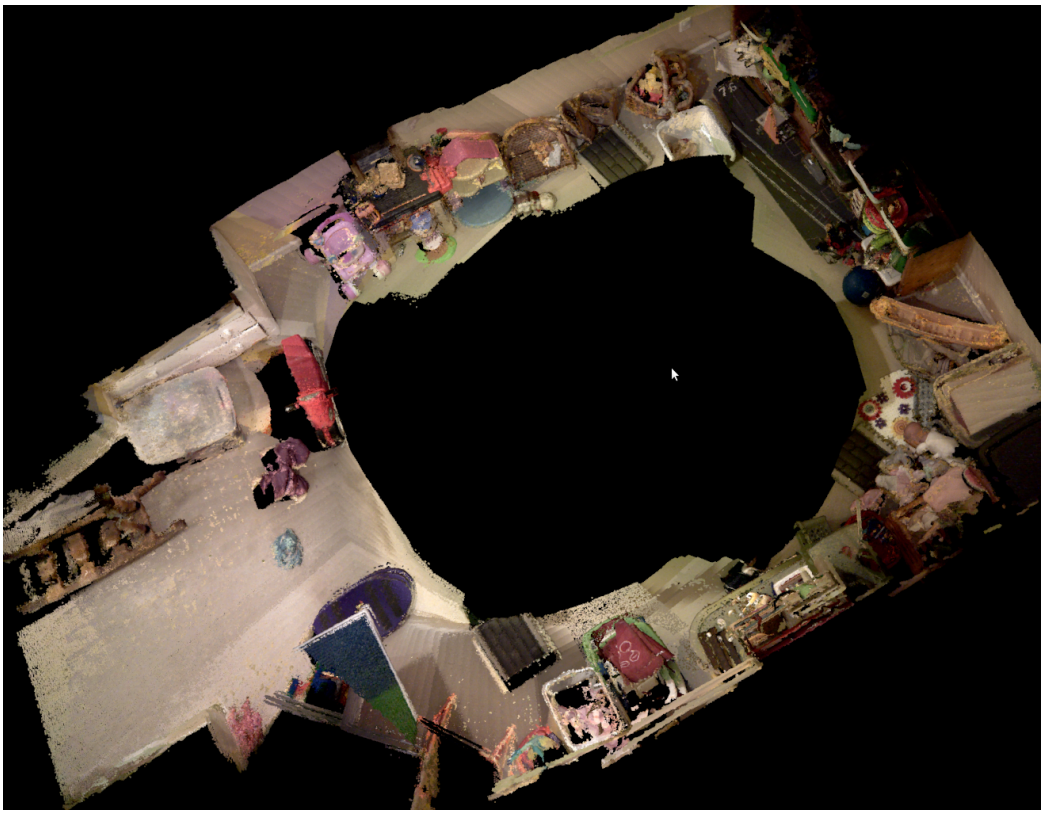


Figure 6.5 A captured model of a playroom. This model was adjusted after loop detection, but the features didn't completely constrain the closure of the model using the joint dense alignment module. The slight offset can be seen in the bottom left of the image.

Figure 6.6 This series shows several images of cleaning product container from varying points of view, highlighting the detail, including the noise, from a relatively close up scan of a single object.

ways are more advanced than what I have presented in this chapter. However, my experience with these newer algorithms remains the same: they work well in some cases, and fail in others. If we want to create reliable robots to operate within our environments with confidence, then we have more work to do. As a way of concluding this chapter, I would like to present a set of comments and lessons learned that focuses on the challenges still present in the field of visual ego-motion and mapping.

6.4.1 On surfels

Surfels were a good choice of representation, and this choice has withstood the test of time: while many experimental algorithms continue to make use of TSDF volumes, there are several new algorithms that have made good use of surfels as well [10], [66], [164], [185]. The surfel maps provide additional information that is exploited during integration time, and they do not require memory to store empty space (as opposed to TSDF volumes).

However, we ran into several difficulties during the implementation of the algorithm. The first challenge was finding the surfels in memory that were relevant to the incoming query frame. To support *continuously* adding surfels to handle more detail (e.g., when the camera is closer to the surface), the surfels were stored as an unorganized cloud, and new surfels were appended to the end. However, as the camera moves, we need to select the surfels that we wish to update, i.e., those that are in the estimated query frustum. This requires projecting surfels into the frustum, and then sorting the "active" surfels to the front of the array. This can be done using a parallel partition algorithm on the GPU. Unfortunately, this adds additional overhead before the alignment can even be computed.

Another challenge was the cost of CPU to GPU (and vice versa) memory transfers. An alternative formulation that could help keep relevant surfels handy (as in the first challenge), enable more efficient swapping, but sacrifices the ability to arbitrarily add new surfels to the cloud is to store surfels in 2D organized clouds, corresponding to the image plane. These *keyframe clouds* keep the relevant surfels at hand for a particular frustum, but discard query surfels that move outside of the frustum, and cannot support multiple resolutions (at least without creating a new keyframe cloud).

6.4.2 On high-resolution models

Many algorithms are enabled when you can capture coherent, high-resolution models of the environment. For example, the models may be used as maps for humans, maps for robots and, after some processing, used in navigation and path planning, sources

for segmentation, and sources for object models for learning 3D object and pose recognition. In addition, the corresponding ego-motion can be used for robot odometry. However, due to the difficulty in generating and maintaining the high-resolution maps, the benefit may not be worth the costs. We want robots to understand the environment deeply, and see it as we do, but it is not clear that a high resolution metric model is the *best* way to do that. For example, we don't see a bunch of points or surface elements, we see objects, parts, and the surfaces they have and rest upon. We see some of the properties of objects and surfaces, and can recognize a portion of a surface as a wall if it is planar and has a normal perpendicular to the ground.

So, a high-resolution local model may be useful (and necessary) to extract some of the higher level semantic information present in the environment, but may not be needed for permanent map storage. What then does an intelligent embodied agent do for maps?

6.4.3 On the future

Robots need scalable and reliable mapping, navigation, and understanding of new and existing environments, especially for life-long adaptability and autonomy. We are still looking for a solution that provides all three components. Therefore, I would like to make three suggestions toward this goal based on my experience with this research. First, discretizing space into a hierarchy of regions can help to simplify the problem and allow for scalability. In addition, this approach would relate well to the way humans conceptualize space, and may therefore provide a better representation for common ground between robots and the people they interact with. Second, using higher level landmarks within the map should offer robustness to varying environmental conditions and pave the way for better understanding. For example, if we replace image features with recognized objects and raw point clouds with extracted planes, walls and corners, then it may be easier to recognize a place, even when objects have moved or the lighting differs from the previously experienced lighting condition. Finally, if we can focus more on the topological structure of the environment and less on metric accuracy, the robot could respond more effectively to changes in the world, and

worry less about updating numbers in its spatial memory. This case is particularly important when robots start to remember things for longer periods of time; anyone who has seen a large map undergo loop closure adjustment after graph optimization knows the many values that might need to change. What if individual objects or events the robot experienced in those places were tied to the global pose of the map? Instead, they could be tied to the lowest-level hierarchical regions, which obtain their global context by moving to higher levels in the tree of places, and will usually require *no* local updates, unless the robot directly experiences changes it wishes to record.

The remaining two chapters shift focus from ego-motion and mapping to object-level concerns when observing the environment. Segmentation is often an important component of understanding the environment, since it serves to logically group individual elements (i.e., points or pixels) into larger parts that can make it easier to learn, recognize, or reason about the world. Recognizing objects and their pose is one of the important steps from the previous discussion that will help us move towards better understanding and away from the limitations of working with points and occupancy grids.

Chapter 7

Temporally Consistent Segmentation

7.1 Introduction

Generating a useful segmentation for a point cloud is an important prerequisite for object recognition, grasping, and scene understanding. Much of the existing work has focused on single-frame segmentation in images or RGB-D image pairs, or global segmentations in single point clouds. In robotics applications, however, we have dynamic control over the sensor viewpoint within the environment and a constant stream of data from sensors making the single-frame or single point cloud approach less desirable, especially during navigation or manipulation related tasks where an up-to-date segmentation can be critical. While it may be possible to re-segment the entire cloud at every frame or segment each frame individually, it is not clear how to maintain consistency between these completely different segmentations[1].

We would also like to take advantage of the 3D data produced by the sensor in order to generate segmentations that respect the geometry of the environment. Image-based segmentation algorithms have difficulty distinguishing between two distinct regions if there is not a significant texture, color or intensity difference between them. However, the geometry of the scene can provide significant hints through depth

[1]This chapter reproduced with adaptations from [157].

discontinuities or abrupt changes in surface normal direction.

The goal of segmentation is to partition a representation of the world into useful chunks. The exact meaning of "useful" depends on the context, but usually we wish for different objects to belong to different segments and for a single segment to belong to no more than one object. A *perfect* segmentation, if such a thing existed, would have one segment per object (where the objects are defined by the application). If objects of interest throughout the scene are divided into more than one segment, then we say there is an *over-segmentation* of the scene. In contrast, an *under-segmentation* results if one segment contains portions from more than one object. Unfortunately, segmentation is an ill-posed problem [16], [138], and even humans have a hard time determining or agreeing on the best partitioning of a scene.

Segmentation is often used as a pre-processing step before object recognition and scene parsing, in order to group similar pixels or points together so the next algorithm has a reduction in complexity. In this case, however, we wish to avoid under-segmentation since an object recognition algorithm will interpret the segments as indivisible units; if a unit contains more than one object, the recognition outcome will necessarily be ambiguous (and incorrect). Therefore, since near-perfect segmentations are so hard to obtain, we strive for effective over-segmentations and rely on subsequent algorithms to merge the related segments into whole objects.

In this research, we aim to incrementally generate an over-segmentation of a point cloud as aligned RGB-D keyframes arrive from an ego-motion estimation process. The segmentation produced at time $t+1$ should have the property of temporal consistency: all points belonging to segments from frames $\{1, ..., t\}$ viewable from frame $t + 1$ should maintain the same label. Points in frame $t + 1$ that were previously unseen may extend an existing label or generate a new label, as determined by the similarity metric in use. Since we currently do not consider dynamic scenes, spatial consistency is assumed such that points at one world location in frame t implicitly map to the same location in frame $t + 1$. We accomplish this by computing an estimation of the camera motion in order to align consecutive clouds and propagate the existing segments forward into the new frame. Segments that lie on the boundaries of known

surfaces are candidates for extension using points from the new frame. The union of these *boundary* segments and the new surface points make up the incremental segmentation set: the set of points in frame $t+1$ that must be segmented.

An effective incremental over-segmentation can be used as input to a higher-level scene-understanding algorithm: coalescing individual segments and providing object labels like "wall", "desktop", and "chair" as the data is streamed from the sensor. While these algorithms are much more complicated, a robot needs hypotheses as soon as possible and the incremental but consistent nature of the data stream could allow for more rapid and timely inference compared to a global batch scheme.

The primary contribution of our work is the extension of an existing point cloud segmentation algorithm to support dynamic octree resizing, new point additions and new occupied voxel notifications, as well as a boundary segment definition in order to support consistent over-segmentations of streaming point clouds. We show that the new incremental segmentation runtimes scale much better with model size as compared to the full segmentation, and the incremental segmentation with boundary expansion results in average segment sizes that are closer to those of the global segmentation process than those segments produced without boundary expansion.

In section 7.2 we discuss the point cloud segmentation algorithm we utilize and how we extend it to support temporally consistent operation. Section 7.3 provides a brief description of the environment modeling algorithm we use from the previous chapter. Section 7.4 discusses the results of our experiments, and we finish the paper with some ideas for future research and our conclusion.

7.2 Segmentation approach

We begin by defining the notation and vocabulary of our approach.

Definition 1 *Definition of point cloud segmentation. Given a point cloud model $\mathcal{M}$, a segmentation is a non-empty set of segments $\mathcal{S} \neq \varnothing$, where $c_i \in \mathcal{S}, \quad i = 1, .., N$ are the components of the segmentation. Each c_i is a disjoint subset of the points in $\mathcal{M}$ such that $\mathcal{M} = \cup c_i$.*

Let $\mathcal{S} = \{c_1, \ldots, c_N\}$ be a segmentation for model $\mathcal{M} = \{o_1, \ldots, o_M\}$, where each $o_i \in \mathcal{M}$ is an object of interest. We give the most basic definition for an over-segmentation:

Definition 2 *$\mathcal{S}$ is an over-segmentation when $N > M$ (often much larger) such that each object $o_i \in \mathcal{M}$ is composed of one or more segments c_j.*

In fact, we are not just interested in any over-segmentation, but in one that has high boundary recall and low under-segmentation error. Boundary recall reflects the percentage of segments with borders that correspond to boundaries in the model (i.e. the boundaries between objects we are interested in and other objects or the background). Under-segmentation error is the percentage of segments that contain a portion of a boundary; in other words, a single segment contains two objects. If we intend to use the segmentation for object recognition, it is especially important to minimize the under-segmentation error in order to ensure that the parts or segments composing an object accurately represent its shape and texture properties and do not model objects with features incorrectly included from other objects.

Note that these definitions do not give us any information about how to actually generate a segmentation. While this depends heavily on the specific algorithm, it typically involves both a function that provides a numerical similarity (or equivalently, a distance) between all elements to be segmented (e.g. pixels or points) and a method of determining the best grouping or segmentation given that function. In this work, we extend the voxel cloud connectivity segmentation (VCCS) method[2] to generate incremental over-segmentations. The following sections describe the details of the original algorithm as well as our extension for incremental processing.

7.2.1 Over-segmentation using voxels

We extend the work of Papon et al. [163] and adapt their voxel cloud connectivity segmentation (VCCS) algorithm for incremental operation. Algorithm 5 shows the

[2]The version from the Point Cloud Library (PCL)[180].

Algorithm 5 Voxel cloud connectivity segmentation

```
1: function VCCS(P, O_a, O_s)
2:     initialize O_a with resolution R_v
3:     initialize O_s with resolution R_s
4:     S ← ∅
5:     O_a ← P, O_s ← P
6:     for s ∈ O_s do
7:         p = closest(s, P)
8:         S = S ∪ to_supervoxel(p)
9:     end for
10:    S ← sparse_filter(S)
11:    for j = 1, d do
12:        for c_i ∈ S do
13:            expand(c_i, O_a)
14:        end for
15:        for c_i ∈ S do
16:            recenter(c_i, O_a)
17:        end for
18:    end for
19: end function
```

high-level design of their original algorithm. In the following we reproduce a summary of each component.

Point cloud generation and voxelization

VCCS relies on a voxelized version of a point cloud. The RGB-D sensor used in the original work as well as this work only provides an RGB image $\mathcal{I} : \Omega$ and a range image $\mathcal{R} : \Omega$ at each time step, where $\Omega \in \mathbb{R}^2$. From these data, we generate an organized point cloud $\mathcal{P}$ by back-projecting each image plane coordinate $\mathbf{p} = \{u, v\} \in \Omega$ into the world relative to the sensor frame according to:

$$\mathcal{P}_{\mathbf{p}} = \left\{ \frac{u - c_u}{f_u} \mathcal{R}_{\mathbf{p}}, \frac{v - c_v}{f_v} \mathcal{R}_{\mathbf{p}}, \mathcal{R}_{\mathbf{p}} \right\}.$$

While this is sufficient for segmenting a single RGB-D frame, it does not by itself take into consideration the motion of the sensor within the world. We address this issue in section 7.3.

The cloud $\mathcal{P}$ is voxelized using an octree data structure created with a *voxel resolu-*

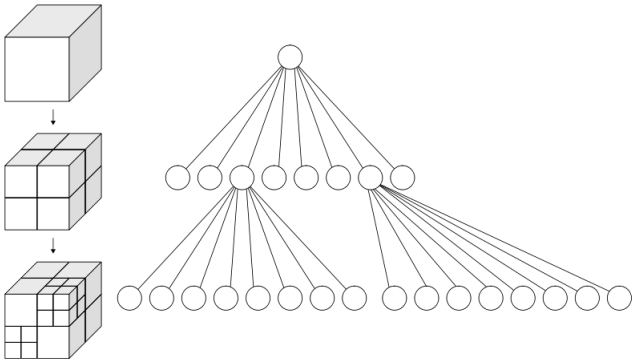


Figure 7.1 The tree structure for a simple octree with three layers. Each node represents a uniformly sized volume in 3D Euclidean space surrounding an origin point o.[3]

tion R_v (see Fig. 7.1). An octree is a spatial data structure, i.e. a data structure that provides some way of subdividing space or objects in space that reflects the relative relationship between the items stored within. The octree structure divides a cubic volume of space into eight octants surrounding an origin $o = (x, y, z)$, corresponding to the following relationships: $\{(< x, < y, < z), (< x, < y, > z), (< x, > y, < z), (< x, > y, > z), (> x, < y, < z), (> x, < y, > z), (> x, > y, < z), (> x, > y, > z)\}$. Each octant may then be recursively divided into its own set of octants, until an individual cell measures R_v on a side. Insertion is very easy, and the organized structure provides facilities for making local spatial searches quick (e.g. find the k nearest neighbors, or all points within a radius). If the points in the cloud are dense enough or the resolution is large enough, multiple points may fall into a single leaf cell. This may be used, as it is in this case, to reduce the complexity of the cloud by effectively subsampling the points.

Adjacency computation

Another simplifying benefit of an octree structure is that it supports straightforward adjacency computation. There are 26 neighbors in 3D space when discretized by axis-aligned cubes. The octree provides a way to check the 26 neighbors for occupancy, and generate an adjacency list for each node.

[3]Figure illustrated by WhiteTimberwolf.

Supervoxel seeding and initialization

The VCCS algorithm utilizes a variant of k-means clustering to generate the segments. Two particularly challenging aspects of the k-means algorithm are choosing k and effectively initializing the cluster centers. The value k determines how many segments are produced and is important since we want to avoid under-segmentation. Once k is chosen, we must distribute the cluster centers throughout the data to begin alternately assigning the points closest to the current center to the cluster and then re-computing the cluster center given points within the cluster. In the current PCL VCCS implementation [163], [179], voxels are assigned to a cluster using a distance function that combines distance metrics related to Euclidean distance, normal difference, and color difference between a candidate voxel and a given cluster. More complex distance functions, such as those mentioned in Papon et al.[163], could include distances between extracted 2D or 3D features at the cost of greater computation time.

While there are many ways to choose k and initialize the centroids, VCCS exploits the following consideration for point clouds: the segments should be regularly distributed over the surfaces in space while attempting to maximize the similarity between points in a single segment. This yields a natural approach for *implicitly* computing k and seeding the supervoxel[4] segments: since we have the points in space representing the world, we can use another application of an octree with a coarser *seed resolution* R_s to find candidates that are reasonably distributed over the occupied space.

Cluster Aggregation

As mentioned in the previous section, cluster aggregation, or associating voxels with the supervoxel cluster centers, proceeds by alternating between two steps:

- (Expansion) For each supervoxel c_i, we iterate over the voxels, adding them to the segment if the distance between the current centroid and the voxel is less

[4]The authors use the term supervoxel to represent a collection of voxels, in the same way superpixel is used to represent a collection of pixels in the image segmentation literature.

than the shortest distance to a center seen by that voxel. All voxels adjacent to this voxel are added to the visit queue.

- (Recentering) The centroid is updated to reflect the mean of the supervoxel

The distance function used here is the weighted L2-norm of the vector of the normalized spatial point to point distance D_s, color distance D_c, and normal distance $D_n = 1 - \langle \nu(p_i), \nu(p_j) \rangle$:

$$D = \sqrt{\lambda \frac{D_s^2}{3R_s^2} + \mu \frac{D_c^2}{m^2} + \epsilon D_n^2}. \tag{7.1}$$

While the original paper states that the centroid stabilizes after an expansion depth of five iterations, the version of the algorithm contributed to the Point Cloud Library (PCL) we have adapted computes the expansion depth with $T = \lfloor \lambda \frac{R_s}{R_v} \rfloor$. This parameter is therefore directly related to the chosen ratio of the seed resolution versus the voxel resolution. The algorithm assumes that each voxel will be tested by at least one cluster. The greater the number of clusters tested for a particular voxel, the greater the competition between the clusters.

7.2.2 Extension for incremental processing

There are two main issues when considering an incremental extension. First, we want to minimize the number of points we have to process, and second, we want to support the temporal consistency property. Therefore, processing each frame independently is undesirable, as mentioned previously. Instead, we want to exploit estimates of the camera motion in order to align the new incoming cloud with the existing cloud we have already segmented.

Assume we have a method of detecting previously unseen points in an incoming frame. Then adding *only* the new points to be segmented generates results like those shown in Fig. 7.2. In that image, a single plane has been incrementally segmented over camera motion, resulting in new point sets for each frame that were *independently* segmented. This yields detectable boundaries between the frame views, since the

Algorithm 6 Incremental VCCS

function INCREMENTAL-VCCS($\mathcal{P}_i$, O_a, O_s, $\mathcal{S}$)
  $B_a \leftarrow \varnothing$, $B_s \leftarrow \varnothing$ ▷ initialize octree observers
  $O_a \leftarrow \mathcal{P}_i$, $O_s \leftarrow \mathcal{P}_i$ ▷ B has *new* voxels
  $\mathcal{S}' \leftarrow \varnothing$
  for $s \in B_s$ **do**
    $p =$ **closest**$(s, \mathcal{P})$
    $\mathcal{S}' = \mathcal{S} \cup$ **to_supervoxel**(p)
  end for
  $\mathcal{S}' \leftarrow$ **sparse_filter**$(\mathcal{S}')$
  $s \leftarrow \varnothing$ ▷ initialize candidate border set
  for $v \in B_a$ **do**
    for $n \in \mathcal{N}_\mathbf{v}$ **do**
      if n has label **then**
        $s \leftarrow$ **supervoxel**(n)
      end if
    end for
  end for
  $s \leftarrow$ **border_filter**(s) ▷ filter border using heuristic
  $\mathcal{S}' \leftarrow \mathcal{S}' \cup s$ ▷ add border to incremental set
  for $j = 1, d$ **do**
    for $c_i \in \mathcal{S}'$ **do**
      expand(c_i, O_a)
    end for
    for $c_i \in \mathcal{S}'$ **do**
      recenter(c_i, O_a)
    end for
  end for
  return $\mathcal{S} \cup \mathcal{S}'$
end function

Figure 7.2 Purely incremental segmentation over multiple frames. This image shows a single plane viewed with camera motion over multiple frames. Note the detectable boundaries between frame views due to lack of segment extension.

segments from the previous frame are not properly extended. In some cases, this may not negatively affect the performance of the segmentation (with respect to boundary recall or undersegmentation error), but could generate more segments than a single global segmentation as well as more segments that are smaller or larger than the average for artificial reasons. Since we would like the *number of segments* and *segment size distribution* to be similar to a single global segmentation of the final model, we must extend the old segments with the new data. We note that segment extension requires a relaxation of the temporal consistency constraint (i.e. instead of "all points" we have "most points"), since extended border segments can ultimately change shape due to centroid updates.

How do we detect the previously unseen points, i.e. the ones that haven't been seen and segmented yet? A general way to do this (independent of the segmentation algorithm) includes projecting the existing labeled points into the new frame and generating a mask used to exclude points in the new frame that correspond to old points. However, since we are manipulating point clouds and have access to spatial data structures, we instead develop a voxel-based approach that exploits properties of the VCCS algorithm and utilizes adjacency lists for boundary detection. Algorithm 6 shows pseudocode for the new method which we describe in more detail in the following sections.

Voxel-based propagation with adjacency search

The voxel based approach is conceptually simple. We extend the existing algorithm to allow for incremental computation by directly tracking newly inserted leaf nodes in the voxel octree data structure and adding support for dynamic octree resizing. New leaf nodes indicate the addition of newly occupied voxels, whereas if a query point maps to an existing leaf node, then that voxel already exists and is occupied in the octree. The latter also implies that the voxel has already been assigned a segment label.

We take advantage of the fact that the existing VCCS algorithm is subdividing the world into voxels and maintaining an octree data structure to manage adjacency

between voxels. We make use of this capability to detect previously unseen points in the query frame and add neighboring border segments using the adjacency list. For each incoming frame, we construct an incremental voxel set, $\mathcal{V}$, that indicates the voxels that must be segmented. This set will also include the previously computed border segments that must be extended with new voxels.

Detecting new points Each query frame point cloud is in world model coordinates, courtesy of the camera motion estimation we perform. This allows us to insert the query points directly into the existing voxel data structure used in the VCCS algorithm. In original form the algorithm did not track or distinguish between newly created voxels or previously occupied voxels. Our modification includes an Observer pattern [68] implementation that allows us to receive notifications for every point insertion that generates a new voxel leaf node in the octree data structure. Since every point must reside at the resolution level of the tree (i.e. the leaves), we can deduce that a point has not been seen before when a leaf is created; otherwise, the leaf contains at least one other point that has been previously seen. Every newly created voxel is added to $\mathcal{V}$, and then queried for border segments.

Boundary detection In order to complete the incremental point set $\mathcal{V}$, we must also detect the existing segments that lie on the boundary in order to determine only those segments we wish to extend using the incoming points. The following definitions will clarify the following discussions.

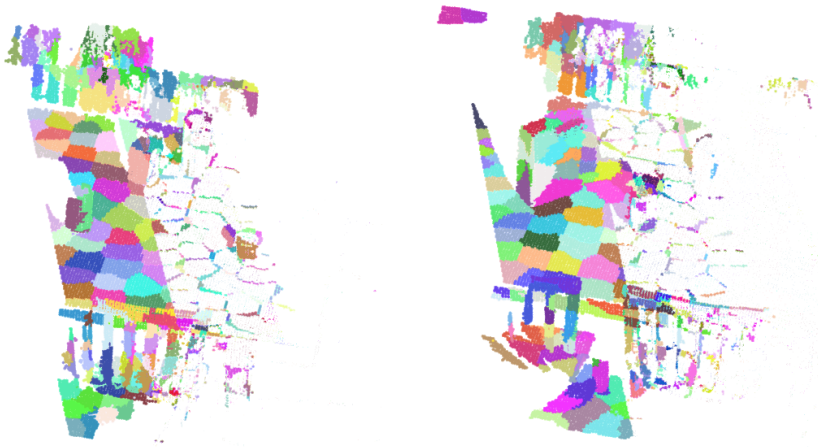
Definition 3 *A segment is an* interior segment *if it is completely bounded by other segments.*

Definition 4 *A segment is a* boundary segment *if it is not an* interior segment.

Since the octree data structure conveniently computes the occupied adjacencies for every voxel in a 26-neighborhood, we exploit this information to search for voxels with an existing segment label. If a new voxel has a neighbor with a label indicating it belongs to an existing segment, then that corresponding segment's points will also be added to $\mathcal{V}$ (since, by the definitions above, the segment is a boundary segment).

(a) No Heuristic, Frame 2-3

(b) No Heuristic, Frame 3-4



(c) Heuristic, Frame 2-3

(d) Heuristic, Frame 3-4

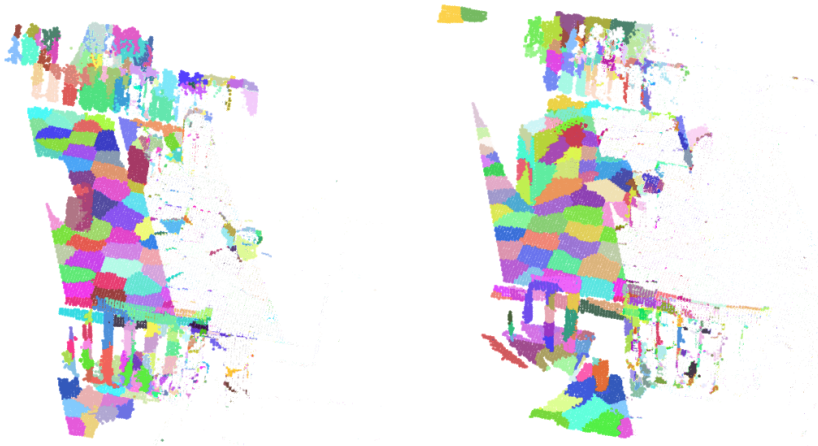


Figure 7.3 Comparison of segment changes, without (7.3a-7.3b) and with (7.3c-7.3d) the boundary heuristic. In the left column, the images show the difference between the segmentation of frames 2-3, and in the right column the difference between frames 3-4. Note that without the heuristic, many more segments have significant differences in their shapes, causing greater perceptual and structural discontinuity between frames.

Boundary heuristic

While the definition of boundary segment correctly handles any case where there exists a new point adjacent to a labeled point, the choice to add the entire segment to the active incremental cloud is not always desirable. In our experiments, we noticed that many interior segments had a few voxels or pixels missing, causing them to be added to $\mathcal{V}$, the incremental voxel set. This often yielded noticeably different segmentations across subsequent frames, for segments that were perceived to be interior and complete (see Figures 7.3a-7.3b).

We address this issue by adding a boundary heuristic: we check the percentage of new unlabeled points or voxels to the total number of occupied adjacent nodes. If the percentage of unlabeled points exceeds a configurable threshold, we consider the adjacent segments as boundary segments, and add them to the incremental list. Otherwise, the new leaf node is simply added to the segment that yields the lowest distance, as in the original expansion algorithm. These segments are not added to the incremental cloud in this iteration, better preserving their original configuration (as seen in Figures 7.3c-7.3d. Note that some of the differences still seen in the previously segmented regions are actually previously unseen points revealed by the camera motion.

7.3 Environment modeling

The segmentation approach we have described relies on good ego-motion and a correspondingly well-aligned model of the environment. We make use of the approach presented in Chapters 5 and 6 to generate models of the environment and then use these to generate fused color and depth images for the incremental segmentation. Ideally, the SLAM algorithm would output new integrated keyframes as the camera moved through the environment, but for the experiments we present in the next section, we simulate that capability after the fact by pre-generating the model and then extracting the color and depth views using the estimated camera motion path.

Figure 7.4 shows sample frames as well as the full models for two of the three scenes we used in the testing dataset.

7.4 Results

We focus on two quantitative metrics to investigate the performance of the incremental versus the global algorithm. We separate the incremental algorithm into two versions with and without boundary expansion. Images from the three datasets used for testing are shown in Fig. 7.4. For the tests, the voxel resolution was set to 8 mm, while the

Figure 7.4 RGB and depth images from the three scenes analyzed.

seed resolution was set to 10 cm.

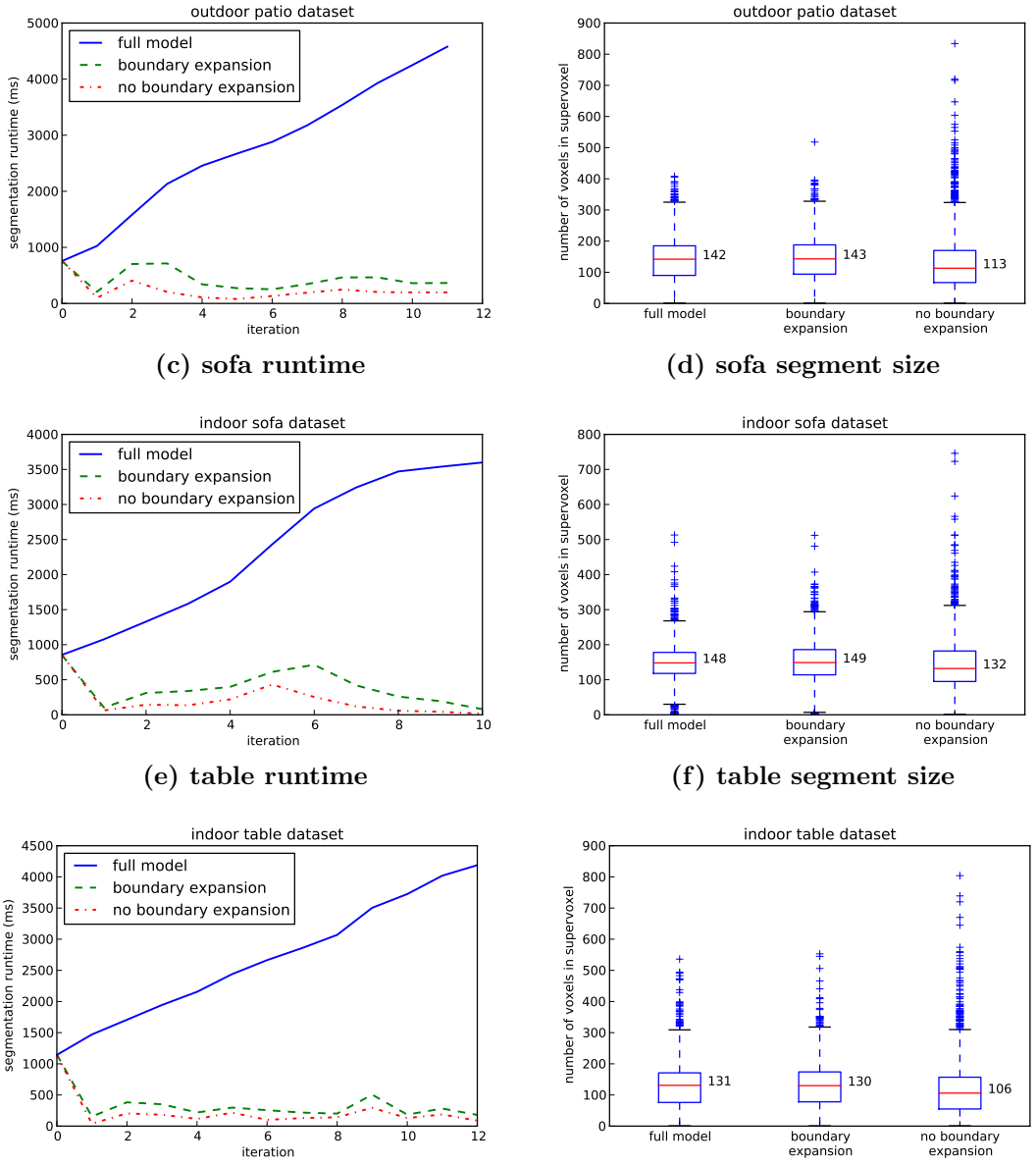


Figure 7.5 Runtimes (7.5a,7.5c,7.5e) and segment sizes (7.5b,7.5d,7.5f) for the three segmentation algorithms run on each dataset. Box plots showing segment sizes in voxels are annotated with the median segment sizes for each algorithm. Runtime plots show that incremental algorithms scale better to increasing model size than full model segmentations. Segment size plots show that segment sizes from an incremental segmentation with boundary expansion more closely match the sizes from a full model segmentation than incremental segmentation without boundary expansion.

7.4.1 Performance

The runtimes of the three methods analyzed are shown in Fig. 7.5. As expected, the runtimes of the incremental processes remain approximately the same as more frames are iteratively added to the model, while the runtime of the full model process increases significantly. The difference in runtime between the incremental processes is due to the boundary identification process and the addition of the boundary voxels to the incremental set. While the runtime is greater than the incremental process without boundary expansion, the boundary expansion method remains lower than even the first frame segmentation.

7.4.2 Segment size distribution

As discussed in Section 7.2.2, incremental segmentation without boundary expansion can lead to prematurely truncated border segments. A byproduct of this truncation is that segmentation clusters may have more or less competition with one another depending on whether they are at the border or not. Ideally, more competition between clusters would increase the likelihood that a voxel would be added to the correct cluster. Also, increased competition naturally regulates the size of a cluster to be less than the maximum allowable size based on the expansion depth, since a portion of the clustering distance metric comes from the euclidean distance to the center of a cluster.

Figure 7.5b, 7.5d, and 7.5f illustrate the sizes of superpixels for the test datasets for each method analyzed. As we expected, segment sizes for the incremental segmentation with boundary expansion mirrors the full model segmentation, while the incremental version without boundary expansion truncates border segments, resulting in consistently lower segment sizes. Counterintuitively, there are also a few large outliers for the incremental method without border expansion. In these cases, truncated border segments are prevented from competing with neighboring segments in the next frame, allowing segments in the next frame to grow larger than they otherwise would.

7.5 Future Work

This research has made positive strides towards an online temporally consistent segmentation of streaming point clouds, but there is still work to be done. As noted in Sec. 7.4.1, the population of the octree takes up time proportional to the size of the input cloud vs. the size of the incremental point set and is not conducive to frame-rate operation. It may be possible to construct a hybrid algorithm utilizing aspects of the voxel-based approach combined with an image-based projection test to minimize the points inserted into the octree. Alternatively, the estimated ego-motion of the sensor may be able to provide hints to the parts of the incoming cloud that are new; i.e. we could take advantage of the image structure of the cloud and begin processing using the edges that are likely to be new. Incorporating this segmentation process into the ego-motion and mapping process on the GPU could provide additional benefits in speed and segment consistency.

While the ego-motion and mapping algorithm handles overlapping point clouds, as well as micro-loops[5], it does not yet handle full loop closures successfully. As is well known in the field of SLAM research, detecting and correcting for loop closures effectively and efficiently is still a work in progress. In this case, assuming the loop closure was detected by the ego-motion estimation algorithm, the segments generated in the recent frames would need to be merged appropriately into the prior segment structure. Detecting the interface boundary and determining how this merge would occur with minimal or no disruption to previous segments requires further research.

7.6 Conclusion

The algorithm described in this chapter addresses the problem of generating temporally consistent incremental segmentations of streaming point clouds with ego-motion

[5]Micro-loops are similar to the standard loop closing problem but are caused by the camera returning to regions recently covered; e.g. imagine a camera sweeping a room in one direction, and then changing direction to sweep over the room recently observed. In this case, it is not possible to assume that all points observed by the sensor relative to only the previous few frames have not already been seen.

estimates derived from the environment modeling algorithm we discussed in Chapter 6. An over-segmentation for the initial frame is generated using a modified version of the VCCS algorithm, which we then extend incrementally by computing the boundaries of the existing segmentation and generate a new segmentation set given the previously unseen points. While the algorithm is not yet suitable for operation at frame-rate, it is clearly more efficient than simply running the algorithm on the composite point cloud at each frame. In addition, due to the boundary extension methodology, the number of segments compare favorably to the global segmentation with similar segment size distributions, indicating that the incremental algorithm is producing near-equivalent results.

Chapter 8

PartNet

The long-term goal of my research driving the topics in this dissertation is the creation of a set of algorithms that enable semantic environment understanding for robots. What is semantic environment understanding? I believe for robots to be effective collaborators, they must (1) know where they are at all times, (2) recognize places they have been, (3) recognize places they have not been (but understand what is familiar about them), and (4) recognize objects in the environment, and their configurations and relationships with respect to each other.

Humans have an amazing capability for robust place and object recognition along with the ability to quickly learn and become comfortable with our surroundings. Not only does this occur in our frequently visited environments, it also occurs in places that we are visiting for the first time as well as places we have not been for a long period of time. Upon first exposure, humans are able to become familiar and gain a functional understanding of the new environment that is almost instantaneous.

For example, we will not run into walls or run into objects that might be in our way, and we also can recall semantic structure, such as "where is John's room," "where is Mary's office," "where is the bathroom," "where are the stairs?" Even out of doors, we will recall the doors on the outside of a building or, analogously in an unfamiliar state park, we will quickly learn the locations of the picnic, fishing, and camping sites. These capabilities are driven by our ability to create cognitive maps using landmarks as well as our knowledge of objects and their meaning, and their relationships to each

other.

Therefore, it is one of my core beliefs that you cannot have an autonomous embodied system exist within an environment and behave intelligently without having these kinds of capabilities; i.e., without having the capability of understanding the environment, being able to learn about it quickly, and then adapt on the fly.

Putting together what we have so far (in this dissertation) means we can build maps of the environment and at the same time, know where we are within that map, at least locally (eventually, globally). Now we need to know about objects.

How do we understand and construct representations of objects within the environment? First, what do we need to know about objects? At the very least, we would like to know what they are, which is the task of object detection and recognition, and we would like to know where they are, which is the task of localization. Classic object recognition systems typically provide a single class label, i.e., "that image or bounding box contains a chair." However, we may want to actually know more information about that particular chair, because we plan to sit on it, or use it to get someplace we cannot reach. Does it have armrests for our comfort if we sit in it? Does it have wheels, implying it will likely move if we try to step on it? Do we need to first find and manipulate the chair to get it into a usable position?

To be able to answer these questions, we need both the pose of objects and information about their parts. Knowing an object's pose is important for a variety of situations, foremost including tasks that would require manipulation of that object, or detecting changes within the environment. Describing the pose of an object has some subtleties, but is relatively straightforward compared to describing the more refined aspects of an object.

Understanding semantic information about particular aspects of an object will allow us to reason better about it and what we can do with it. Is that a high chair or a low chair? Does it have arms? Can it be moved easily? Is the door handle a lever or a knob? One might consider this solvable through a more fine-grained classification approach, but that leads to a combinatorial explosion of aspects that may best be avoided by recognizing properties individually. This is not just about fine grained

classification, we cannot really represent all the possible states of all objects in the world as *specific classes*. However, there *is* additional information, and we would like to know about it. We want to be able to reason about it. This leads us to parts.

Knowing object parts and understanding the relationship between parts helps address this combinatorial explosion of states that we might encounter in the world and be able to reasonably determine what actions we need to take to achieve whatever specific the goal we have at that time. Learning to segment and recognize parts will give us this ability to chunk the world into meaningful components at a finer granularity.

In this chapter, we look at the feasibility of training a deep convolutional neural network to recognize parts within a given class. In the following sections, I describe a network I call *PartNet* that I apply to the task of detecting and recognizing the parts within a whole. Since I have mentioned chairs several times, and there is a sufficient number of chair CAD models to select from to make it a feasible dataset for deep learning experimentation, I collect a randomly sampled set of chair models, and generate a partially hand-annotated data set for the purposes of supervised training. While I am ultimately interested in developing a learning system for object parts and components in an unsupervised setting, it is useful to approach that difficult task by finding out what works and does not work in a supervised setting.

8.1 Parts

We define a part $\mathcal{P} \in \mathcal{O}$ as a functional subvolume of an object, which itself may be an object. This is general enough to capture multiple interpretations of an object "part," such as when we mean a literal subset of an object's surface that serves a semantic purpose. For example, many office chairs have armrests. Many of those armrests could literally be detached from the rest of the chair and are themselves objects; these separable objects are adjustable and integrated into the chair to serve the armrest purpose. In contrast, other chairs, whether inside or outside an office environment, may have armrests that are not obviously detachable from the whole.

However, that part of the object clearly *affords* being used as an armrest[1], therefore we want to recognize that it serves that role.

Parts have both a true hierarchical relationship with an object, in that an object may be composed of separate components that are put together to make a whole, as well as a functional relationship to the object. In many cases, you can take the object apart into its components. They may or may not be particularly useful on their own, but they contribute to the overall structure and function of the whole. The definition of part *also* represents the recognition that there may be specific functions for a non-detacheable portion of an object, meaning that a portion of the object is used for a particular task; this is affordance oriented and it may or may not coincide with the hierarchical compositional view of parts. One may argue at length about the philosophical nature of this question between objects and their parts and the segmentation between them (in fact, someone does [22]); there is likely no single answer or definition that would satisfy everybody, and cover all cases.

In this dissertation, we apply the dual definition with an emphasis on the semantic use of a portion of an object.

8.2 Problem

We would like to recognize aobject parts and estimate pose using data from sensors that may be mounted on an embodied intelligence system. This includes color images, color and depth, or color with depth and normals; i.e., images $I_i \in \mathcal{I}$ where each image is $H \times W \times D$ where $D \in \{3, 4, 7\}$, for RGB, RGB-D, and RGB-D-N, respectively. Given a single I_i with a known object class (e.g., this could be the output of a previous object detection and localization task) the goal is to detect and localize the visible object parts and estimate the pose of the object. In the following, we formulate the part detection as an instance segmentation task. Given an image of

[1]I thought it would be interesting to look up the definition of an armrest in Google's dictionary, and it provided "a padded or upholstered arm of a chair or other seat on which a sitter's arm can comfortably rest." It's funny how ill-defined our notions of parts (and perhaps our use of language), even as prevalent as they are. So, if the armrest is not padded or upholstered, it's not actually an armrest?

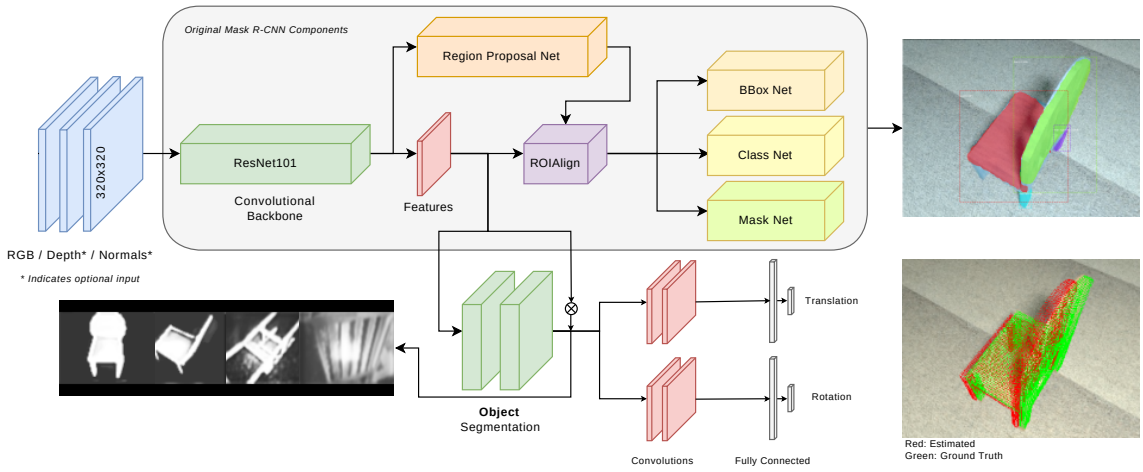


Figure 8.1 The PartNet architecture. The core architecture of the system is shared with Mask R-CNN[82]. We split out a subnetwork to handle the pose estimation, which begins with a segmentation component, and then feeds to a bifurcated network that generates the translation and rotation estimates. Depending on the network configuration, output from the part mask generation is concatenated with the feature maps and segmentation before being fed to the pose network.

an object, we would like to label the pixels for each instance of an object's parts with the corresponding part class. We denote the instance segmentation labeling as $\mathcal{L} = \mathbb{R}^{H \times W \times E} \times \mathbb{N}^{E}$. This gives us localization information for each detected part. Therefore, the function $\Psi_m(I_i; \theta) \rightarrow \mathcal{L}$ produces a set of mask images of cardinality $|E|$ and a corresponding label for each mask.

The pose estimation task produces a rotation and translation for the object in the camera's frame. Therefore, this network, $\Psi_p(I_i, \mathcal{L}*; \theta) \rightarrow T_c^o$, where $T_c^o \in SE(3)$ and refers to the pose of the object in the camera frame. Note that the function optionally takes as input the mask and class output provided by the previous function.

8.3 Approach

We instantiate the functions Ψ_m and Ψ_p as convolutional neural networks. To avoid starting from scratch and because the part segmentation tasks are similar, we start from the Mask R-CNN network of He et al. [82]. We use the ResNet101 backbone and only make one small modification to allow the network to accept arbitrary image channels to handle the additions of depth and normals.

Figure 8.1 illustrates a simplified block diagram of the overall architecture, showing the data flow between the existing structure of the Mask R-CNN (MRCNN), as well as the new additions for pose estimation.

8.3.1 Supervised part detection

Parts have a relatively long history[2], if we include human pose segmentation and pose estimation. Human "parts" consist of our articulated body parts, arms and legs and hands and feet, and fall squarely into the "functional subvolume of a whole" definition of parts. Work on pictorial structures by Felzenszwalb and Huttenlocher [56] (who applied the part detection and model construction for human pose) was itself based on the work or Fischler and Elschlager [62], who apply parts and a deformable model to face detection. Deformable part models [52], [53] are the more recent extension to these concepts. Without going into great detail on the operation of these algorithms, I want to emphasize an important point: outside of the earlier models that specifically modeled body parts, the recent part models do not emphasize **semantic** parts. Instead, they aim to learn a latent representation of parts that are formulated to best discriminate the object from the background.

Why do most learning algorithms now focus on latent representations as opposed to explicit semantic parts? Primarily due to a lack of data! We were initially inspired by the work of Tulsiani et al. [203] in learning to extract 3D primitives from volumetric representations of objects. They use a REINFORCE-based [218] algorithm to train a network to regress primitive parameters for cuboid volumes that best fit the object according to a pair of loss functions that induce the primitives to align to the surface. I am as enthusiastic as the authors are about the potential for new investigation into building simplified representations of objects, similar to the work of Nevatia and Binford [151] and Biederman [14]. However, unfortunately for our robots, they don't see the world as complete, axis-aligned, volumetric objects. Instead, they only see snapshots and partial views (albeit often with depth). We do not expect these data to produce the same results.

[2]At least in "computer vision" timescale.

While I believe that perceiving simplified versions of objects and recognizing parts without direct supervision is where we are heading, in this dissertation I focus on the first step: given a set of semantically labeled objects within a single class, can we get a network to extract the parts? We can build on this foundation in the future (I discuss possible routes to do this in the next chapter).

The MRCNN network provides the instance segmentation capability. We provide the network with a set of classes that correspond to *parts* of a single object class, and the network is trained to output a set of bounding boxes, masks, and the classes for each of the box/mask pairs. It does this by training a subnetwork to evaluate region proposals using a region of interest feature extraction mechanism called ROIAlign. Since the features are only computed once, the process is very efficient, and allows the network along with the attached "heads" (class, bounding box, and mask) to be trained end-to-end. Our version of the MRCNN network uses the ResNet-101 [83] network as the feature generating backbone, and implements feature pyramid networks [130] for feature extraction. For more information on the details and the progression from the R-CNN to the MRCNN, see **girshick_fast_2015**, [72], [82], [130], [174].

The use of MRCNN is not particularly interesting in and of itself for this research. What is interesting is that we show that the instance segmentation approach is applicable to part labeling, and performs well on images when paired with depth maps.

8.3.2 Pose Estimation

The MRCNN network does not support pose estimation. We extend the original network by creating a new head that is composed of three primary components: an object segmentation phase, followed by the bifurcated regression network for rotation and translation. We instantiate three instances of the network to investigate the effects of the part detection on the pose estimation process, as well as the contribution between two different rotation regression formats.

Figure 8.2 describes the first addition to the network to support pose estimation. The segmentation is currently used to generate bounding boxes that define region of

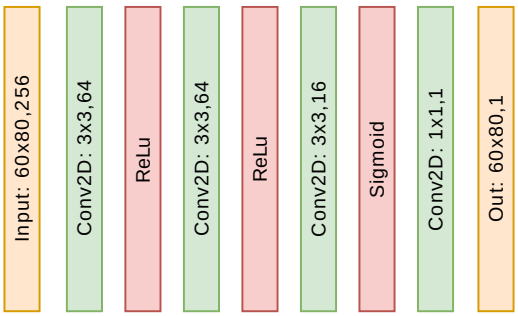


Figure 8.2 The segmentation network layers, listed left to right.

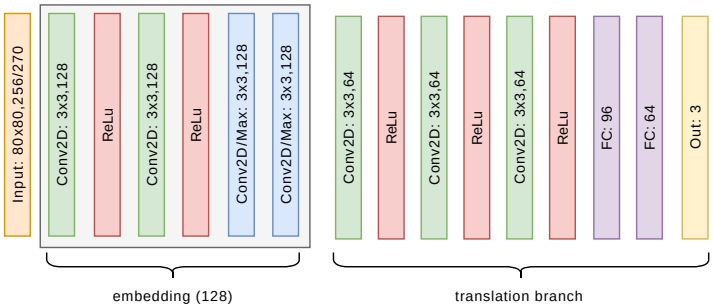


Figure 8.3 The translation branch layers of the pose network.

interests (ROIs) for extracting features from the feature pyramid network [130]. The size of the feature map we extract is a hyperparameter, and we set it to $[80, 80]$ here to accommodate the optional feature masks we use for the "+Parts" variants.

The features extracted using the segmentation are then passed to regression networks for translation and rotation. Shown in figures 8.3 and 8.4, the networks are very similar: both branches pass the features from the feature pyramid network (FPN) through a small encoding network before processing them through two stages (common in many regression networks targeting pose): a set of convolutional layers followed by a set of fully connected layers and terminating in the appropriate sized vector output. We utilize rectified linear unit (ReLU) activation throughout until the final layer, which allows is a linear activation to account for negative values.

Similar to PoseCNN [222] and the Riemannian CNN from Mahendran et al. [135], but in contrast to Gupta [77], [78], Grabner and Lepetit [74], and Wu et al. [221], our pose estimation directly outputs the translation and rotation parameters for the object in the camera frame. Other approaches output additional information such as an object wireframe (as in [221]), or may require additional steps, such as ICP

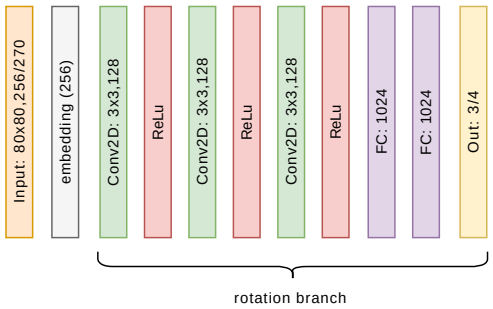
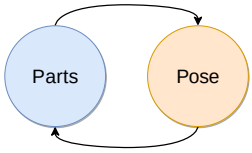


Figure 8.4 The rotation branch layers of the pose network.

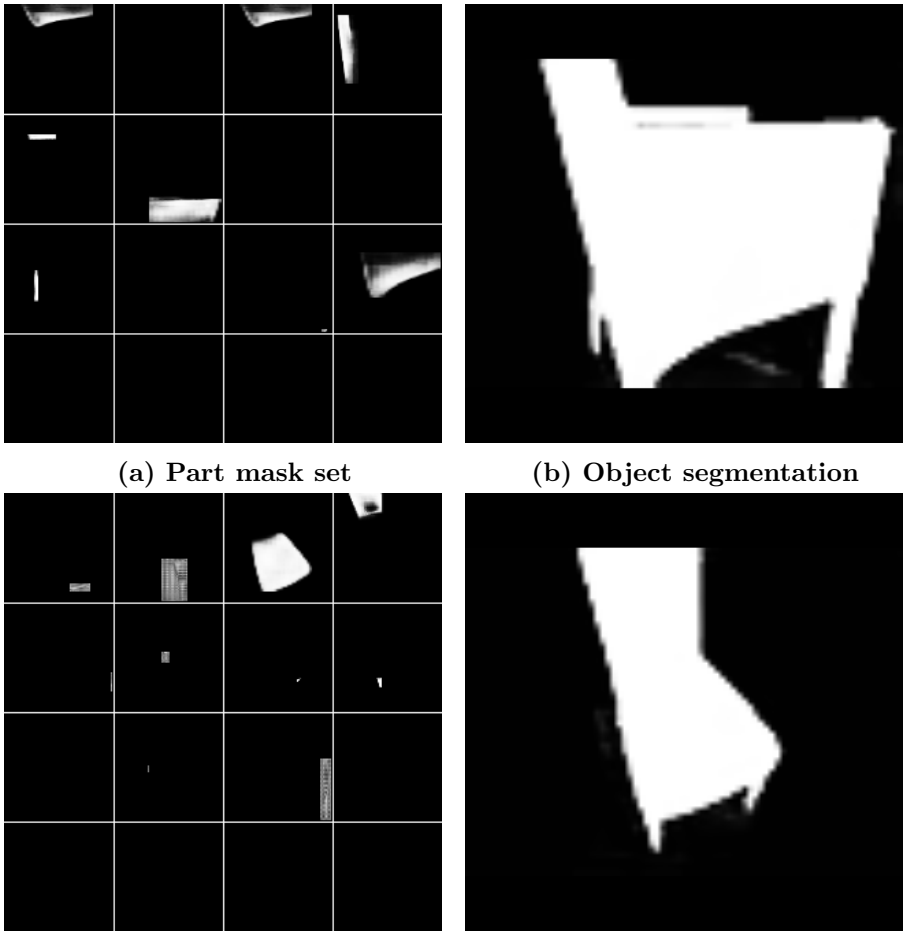
fitting after coarse pose estimation [77], [78], or solve a perspective-n-point (PnP) from estimated object bounding box points [74].

Part contributions



I believe that recognizing parts and their location can help in estimating the pose and that similarly, knowing the pose of the object can provide information about where to look for specific parts. While such a circular and iterative process may yield better results than we can achieve otherwise, in this work we examine a single arc of that graph: whether providing information about the detected parts to the pose network improves pose estimation accuracy.

There are several ways we could pass this information: (1) a vector of parameters that describe the bounding boxes and corresponding class values, (2) a derived graph structure emphasizing neighborhood relationships (perhaps something like the latent graph in [88]), or (3) the set of masks indicating shapes, location and class type. We select the last item for this work, as the network is already producing the masks (in one form) and it is straightforward to append the mask information as simple additional channels in the ROI feature tensor. However, I suspect that item (2) may be part of an effective way to include additional reasoning about object structure into the system. I consider the mask-append method the baseline approach. Figure 8.5 shows an example of the part masks and segmentation masks generated by the network.



(a) Part mask set **(b) Object segmentation**

(c) Part mask set **(d) Object segmentation**

Figure 8.5 Examples of part masks generated for the pose network. A multi-channel tensor representing each part class was reshaped into a grayscale image by tiling the 13 part classes into a 4x4 array.

We hypothesize that information about the presence and spatial location of the parts in the camera frame as provided by the masks will improve the pose estimation accuracy.

Pose representation

Representing rotations in a deep learning framework such as a convolutional neural network is a challenge. The first choice is whether to regress the pose values directly or to implement classification of the pose. For each option, there are several more choices. There are 3 options to consider covering the regression approach: Euler angles, quaternions, and rotation vectors from the Lie algebra $\mathbf{so}(3)$. We'll briefly

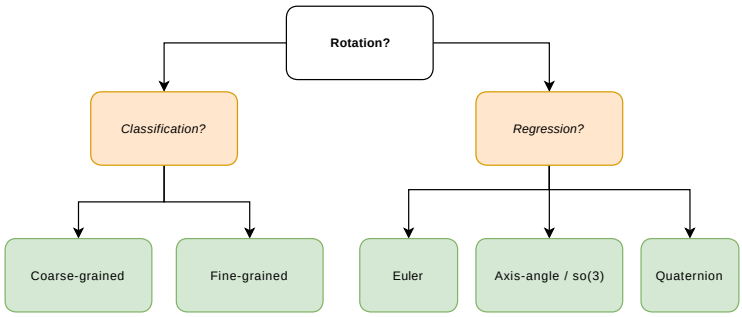


Figure 8.6 The rotation representation hierarchy.

discuss each of these approaches in turn.

In pose classification, we discretize the pose space, often just rotations, into a set of classes where each class represents a continuous range of poses. Coarse-grained classification discretizes the space into few bins, while fine-grained classification allows for more accuracy at the cost of more complexity in the representation. For rotation in a single plane (e.g., estimating an object's yaw in the camera frame), it is common to bin around the compass points in a coarse formulation, while a fine approach might bin for every degree.

While discretizing the range of a single axis rotation is straightforward, it is not obvious how to discretize SO(3) in full; typical approaches often discretize a spherical rotation encoding: using elevation, azimuth, and in-plane rotations. Discretizing the Euler angles is another possibility, although this admits the same problems as in other uses of Euler angles (i.e., gimbal lock and multiple conventions).

In this dissertation, we limit our investigation to quaternions and rotation vectors in $\mathbf{so}(3)$.

Quaternions are a popular 4D representation for rotation and orientations in 3D. Although unit quaternions do provide a double cover of $SO(3)$ (i.e., q and $-q$ represent the same rotation, in a similar way that a 45 degree rotation in the plane represents the same ending point as -315 degree, although the angles are clearly different). Quaternions can also be used to interpolate between two different orientations (i.e., spherical linear interpolation, or slerp) since mathematically, the difference between the two rotations forms a geodesic on the four dimensional unit hypersphere. While quaternions have these benefits, only unit quaternions represent rotations, which means any

mechanism we use must either try to enforce the normalization during learning or accept non-normalized quaternions and normalize them before use. In practice, we end up doing both because the network does not consistently produce normalized quaternions.

Rotation vectors in so(3). Rotation vectors are a more compact encoding (3 vs. 4 values) than quaternions and do not require normalization. In fact, they are specifically un-normalized vectors that encode an axis-angle rotation: $(\hat{\mathbf{v}}, \theta)$, where θ is the amount of rotation around the unit vector axis $\hat{\mathbf{v}}$. The encoding is accomplished by scaling the axis by the angle: $\theta\hat{\mathbf{v}}$. Rotation vectors are derived from the Lie algebra for Lie group $SO(3)$.

Loss function

To train the PartNet to generate pose estimates, we must provide a loss function that computes the difference between the network's estimation of the pose and the actual ground truth pose. As shown in the architecture diagram in Fig. 8.1, the network outputs the loss as 6 or 7 output neurons: 3 neurons for translation in the translation branch, and 3 or 4 neurons in the rotation branch, depending on whether the rotation is encoded as a rotation vector in $\mathbf{so}(3)$ or as a quaternion. We formulate the pose loss as the following:

$$\mathcal{L}_{\text{pose}} = \frac{1}{B}\sum_{i=0}^{B}\left[\mathcal{L}_{\text{trans}}(t_i, \hat{t}_i) + \mathcal{L}_{\text{rot}}(\mathbf{R}_i, \hat{\mathbf{R}}_i) + \mathcal{L}_{\text{reg}}\right]. \tag{8.1}$$

Translation loss. The translation loss $\mathcal{L}_{\text{trans}}(t_i, \hat{t}_i)$ is implemented in a straightforward manner as the squared Euclidean distance: $\left\|t_i - \hat{t}_i\right\|_2^2$.

Rotation loss. The rotation loss is not as straightforward. Since $\mathbf{R} \in SO(3)$, the set of 3×3 orthogonal matrices $\{\, \mathbf{R} \mid \mathbf{R} \in \mathbb{R}^{3\times3},\ \mathbf{R}^T = \mathbf{R}^{-1},\ \det(\mathbf{R}) = 1 \,\}$ representing rotations in $\mathbb{R}^3$, we require a loss function that computes the geodesic distances between rotations, i.e. the length of the shortest path on the group manifold.

The geodesic distance for elements of $SO(3)$ is given in:

$$d(\mathbf{R}_1, \mathbf{R}_2) = \left\| \ln(\mathbf{R}_1^{-1}\mathbf{R}_2) \right\|, \tag{8.2}$$

where $\ln(\mathbf{R})$ is the logarithmic map of $SO(3)$ that takes elements of $\mathbf{R}$ to a skew-symmetric matrix $\omega \in \mathbf{so}(3)$.

Since we would like to avoid the 9-element overparameterization of rotations in $SO(3)$, we instead parameterize our rotations with 3 or 4 elements. For the default implementation, we chose the 4-element quaternions, and translate the distance metric above into the following formula:

$$d(q_1, q_2) = 2\left|\ln(q_1 q_2^*)\right|, \tag{8.3}$$

where $q_i \in \mathbb{S}^3$ are unit quaternions, which form the unit hypersphere as a subset of $\mathbb{R}^4$. Before describing the quaternion logarithm, first recall Euler's formula:

$$e^{i\theta} = \cos\theta + i\sin\theta. \tag{8.4}$$

Glossing over background details, we note that unit quaternions can be thought of as a generalization of imaginary numbers. Let $\mathrm{Im}(q) = \frac{q-q^*}{2}$ which can be seen to yield $\mathbf{q}_v$, the imaginary *vector* part of the quaternion. If we let $\mathbf{h} = \frac{\mathrm{Im}(q)}{|\mathrm{Im}(q)|}$, then $\mathbf{h}^2 = -1$, since

$$\mathbf{h}^2 = \left(\frac{\mathrm{Im}(q)}{|\mathrm{Im}(q)|}\right)^2 \tag{8.5}$$

$$= \left(\frac{\mathbf{q}_v}{|\mathbf{q}_v|}\right)^2 \tag{8.6}$$

$$= \hat{\mathbf{q}}_v^2 \tag{8.7}$$

$$= [x\mathbf{i}\ y\mathbf{j}\ z\mathbf{k}]^T[x\mathbf{i}\ y\mathbf{j}\ z\mathbf{k}] \tag{8.8}$$

$$= (x\mathbf{i})^2 + (y\mathbf{j})^2 + (z\mathbf{k})^2 \tag{8.9}$$

$$= -x^2 - y^2 - z^2 \tag{8.10}$$

$$= -(x^2 + y^2 + z^2) \tag{8.11}$$

$$= -1 \tag{8.12}$$

Then, a quaternion can be expressed in polar form using Euler's formula as:

$$e^{\mathbf{h}\frac{\theta}{2}} = \cos\frac{\theta}{2} + \mathbf{h}\sin\frac{\theta}{2}. \tag{8.13}$$

Since the quaternions provide a double cover of $SO(3)$, we must use half the desired angle. The quaternion logarithm is then:

$$\ln\left(e^{\mathbf{h}\frac{\theta}{2}}\right) = \mathbf{h}\frac{\theta}{2} \tag{8.14}$$

Which shows the relationship between quaternions and rotation vectors, since $\mathbf{h}$ represents the unit vector axis of rotation, and $\frac{\theta}{2}$ is half the angle of rotation around the axis. Therefore, the map $\mathbb{H} \to \mathbf{so}(3)$ is defined as $q \mapsto 2\ln q$. We also see how Eq. 8.3 works: $q_1 q_2^*$ computes the *difference* between the quaternions, the log computes the half rotation vector which we then use to find twice the magnitude to yield the angle between the original quaternions.

In practice, we must be slightly careful when we are trying to compute the *minimal* angle between the quaternions. Due to the double cover, we must orient the quaternions in the same hemisphere of the hypersphere; i.e., we may need to negate one of the quaternions. We check this efficiently by computing the magnitude of the

chord between the points on the hypersphere $|q_1 - q_2|$; if it is greater than $\sqrt{2}$ we flip the sign of one of the quaternions.

Since the network does not implicitly "know" that quaternions must be unit vectors in $\mathbb{R}^4$, we must also add in a quaternion regularization term that penalizes the network for producing non-normalized quaternions. Therefore, the full loss $\mathcal{L}_{\text{rot}}$ for quaternions is:

$$\mathcal{L}_{\text{rot}}(q, \hat{q}) = 2 \left| \ln(\frac{q}{\|q\|} \hat{q}^*) \right| + [\|q\| - 1]^2 \tag{8.15}$$

When the network is predicting rotation vectors, we do not use the norm of the difference between the vectors, nor do we use the exponential map to compute the distance as in Eq. 8.2. Instead, we use the slightly more efficient conversion from rotation vectors to quaternions, and use the same distance function (Eq. 8.3). To convert from rotation vectors $\mathbf{v} = \mathbf{h}\theta$ to quaternions, we use the following:

$$\theta = \|\mathbf{v}\|, \ \mathbf{h} = \frac{\mathbf{v}}{\theta}, \ q = \left[\cos\frac{\theta}{2} \quad \mathbf{h} \sin\frac{\theta}{2} \right] \tag{8.16}$$

However, in this case, we no longer need the regularization term in Eq. 8.15, since the conversion generates unit quaternions by construction.

Regularization loss. This term implements the standard weight decay procedure implemented for most neural network training algorithms; i.e., weights are multiplied by a factor less than 1. This is thought to improve generalization performance, especially with smaller data sets and/or large parameter sets. However, according to a paper under review for ICLR 2019 [4], they found for first order network optimizers such as Adam, the effect is accounted for through an effective increase in learning rate.

8.3.3 Data Generation

Generating and annotating data is a significant time investment. Since we were interested in investigating a simplified supervised learning approach as an initial baseline, we searched for real and/or synthetic data that would satisfy our requirements: ob-

Figure 8.7 Exemplar SceneNet RGB-D frames. Images reproduced from [139] ©2017 IEEE.

jects in a relatively non-cluttered environment with parts and pose annotations. While we found some datasets that fit a portion of the requirements (Sun RGB-D [190], SceneNN [98], and ScanNet [37]; mostly providing scenes with objects and pose), we did not find any with all the requirements.

Based on our previous experience in capturing and annotating clean datasets from real environments, we chose to pursue a synthetic scene generation approach, with the assumption that the synthetic learning could later be merged with real data. ShapeNet is a large database of CAD models [23], and the infrastructure of SceneNet RGB-D provides a set of software tools to generate and render pseudo-realistic environments with objects from ShapeNet using a photorealistic renderer based on photon mapping [139]. Figure 8.7 shows several example renderings from the original SceneNet dataset.

We made two modifications to the SceneNet RGB-D scene and trajectory generators. We first modified the scene generation parameters to choose a single object from a category of our choosing (i.e., chairs in this case), and to place it near the origin of the randomly selected environment. We also modified the environment models to place the origin at a sufficiently central location, to allow for as much camera movement as possible.

Secondly, we augmented the SceneNet RGB-D trajectory generation code to support "spherical" trajectories where our target object remains fixed at the camera look-at origin, and the camera moves on the surface of a sphere with a smooth time-

Figure 8.8 Example chair renderings from our dataset.

varying radius throughout the trajectory. This provides a trajectory that effectively moves the camera along a path normal to the surface of the sphere for dollying the camera near and far from the object, while the motion on the sphere provides views of the object (and the surrounding environment) from various orientations. As in the original work, we keep the camera pointed "up" during all motion.

Figure 8.8 shows several examples of the chair renderings in our dataset. Our training set consists of 62,222 frames with color, depth, normals, part masks, object instance mask, and camera/object pose. The testing set consists of 37,286 frames.

In order to efficiently generate labels for tens of thousands of images, we needed to be as lazy as possible. Therefore, we developed a point cloud annotator built on the Open3D library [235]. The annotator allows us to select multiple points in either polygonal or rectangular regions of infinite depth, based on the camera view frustum. The interface is minimal, using a few key commands and a text-based interface to provide labels to each selected region. We annotated over 150 individual chair models in this way.

While the annotator worked using a point cloud, we needed some way to apply the point labels to the CAD models in order to generate proper depth-aware masks (while a point cloud is a sampled representation of a surface, we needed a real surface to enable OpenGL to do depth buffering). Therefore, we developed another piece

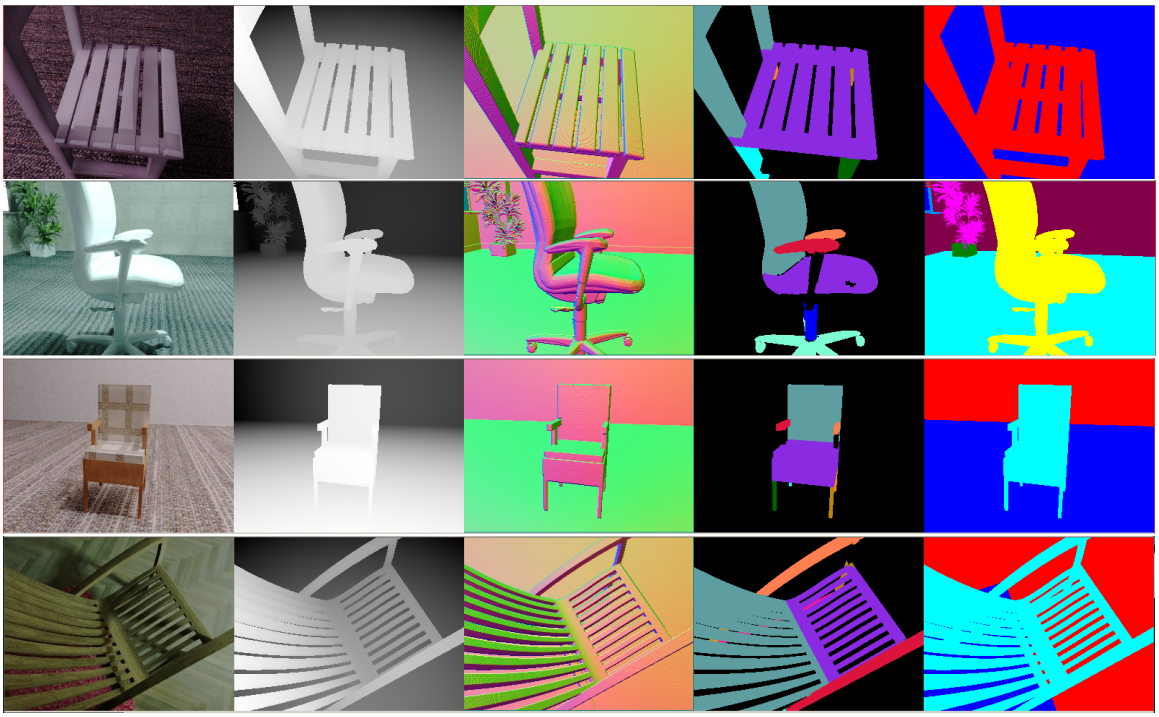


Figure 8.9 Data images contained for each frame in the dataset. In order from left to right: RGB, depth, normals, part masks, scene instance segmentation.

of software that modifies the existing CAD models by associating face colors with the nearest labeled point value. Since many models are represented efficiently with low polygon counts, this required us to subdivide the faces before labeling to reach a suitable vertex resolution.

8.4 Experiments

We run three different network configurations where each configuration produces part detections and object pose estimates. The configurations are as follows:

Pose The pose network is trained to output part masks and object pose but with no part information passed to the pose network head. The features are extracted from the FPN and used without any augmentation.

Pose+Parts In this configuration, we insert a custom layer to compute resized part masks and concatenate them with the incoming feature pyramid. The hypothe-

Parameter	Value
S_1	100
S_2	100
λ_1	0.0005
λ_2	0.005
Steps/epoch	250
Batch size	4
Optimizer	Adam
λ exp. decay	0.95

Table 8.1 Training parameters for our experiments.

sis here is that providing information about the detected parts and their relative locations will provide support to improve the pose estimation.

Pose+Parts+so(3) The network configuration here is the same as in Pose+Parts, but we investigate an alternate encoding for the orientation estimate. As described in section 8.3.2, the original network output quaternions. Since quaternions require an additional component in the loss function to help keep the output normalized, we wanted to investigate the **so**(3) rotation vector. This network uses a modified pose loss function to train the rotation network to output rotation vectors.

For each network configuration, we run 4 different input modalities, corresponding to RGB, RGB-D, RGB-N, and RGB-D-N.

8.4.1 Training

Each MRCNN network is initialized with ResNet-50 weights pre-trained on ImageNet [40], while the remaining layers of the ResNet-101 are intialized randomly. Training proceeds in two stages:

1. The first stage trains the core MRCNN along with the segmentation branch for S_1 micro epochs with learning rate λ_1.

2. The second stage begins where the first stage leaves off, uses a higher learning rate λ_2, and trains the entire network (with the pose head) for an additional S_2 micro epochs.

For a non-exhaustive summary of the summary of the parameters, see Table 8.1.

We train one network per GPU[3] with preferably no restarts[4]. We found that we needed a lower learning rate to avoid generating NaNs in the loss functions for the MRCNN implementation. This issue greatly reduced the learning speed, and could be due to lack of effective weight regularization. In future experiments, this problem warrants further investigation (it seems the weights are disappearing and causing divide-by-zero errors in the class bounding box pipeline).

8.5 Results and Analysis

We report the results of training and evaluating our networks in this section. Testing and metric collection occurred on a 100 element randomly selected subset of the test data.

For reporting on the part detection accuracy, we use the mean average precision metric computed using the standard set of intersection over union (IOU) metrics (0.5-0.95). For our implementation of MRCNN, we specifically use IOU computed over the actual masks, since that is the output of interest.

Pose estimation is reported in a similar manner: we consider an orientation correct if it differed by no more than a given threshold τ_r, which we vary between 0 and 0.51 radians; we consider a translation correct if it differed from the ground truth by no more than a threshold τ_t, which we vary between 0 and 0.5 meters.

Figure 8.10 shows both the good and bad qualitative results for a single network configuration.

[3]Either an NVIDIA P6000 with 24GB or a K6000 with 12GB. While we could train with larger batch sizes on the P6000, we had to choose a batch size that would fit on both GPU models in order to keep the parameters consistent.

[4]The Keras [29] framework makes it challenging to stop and restart without affecting the continuity of the loss values.

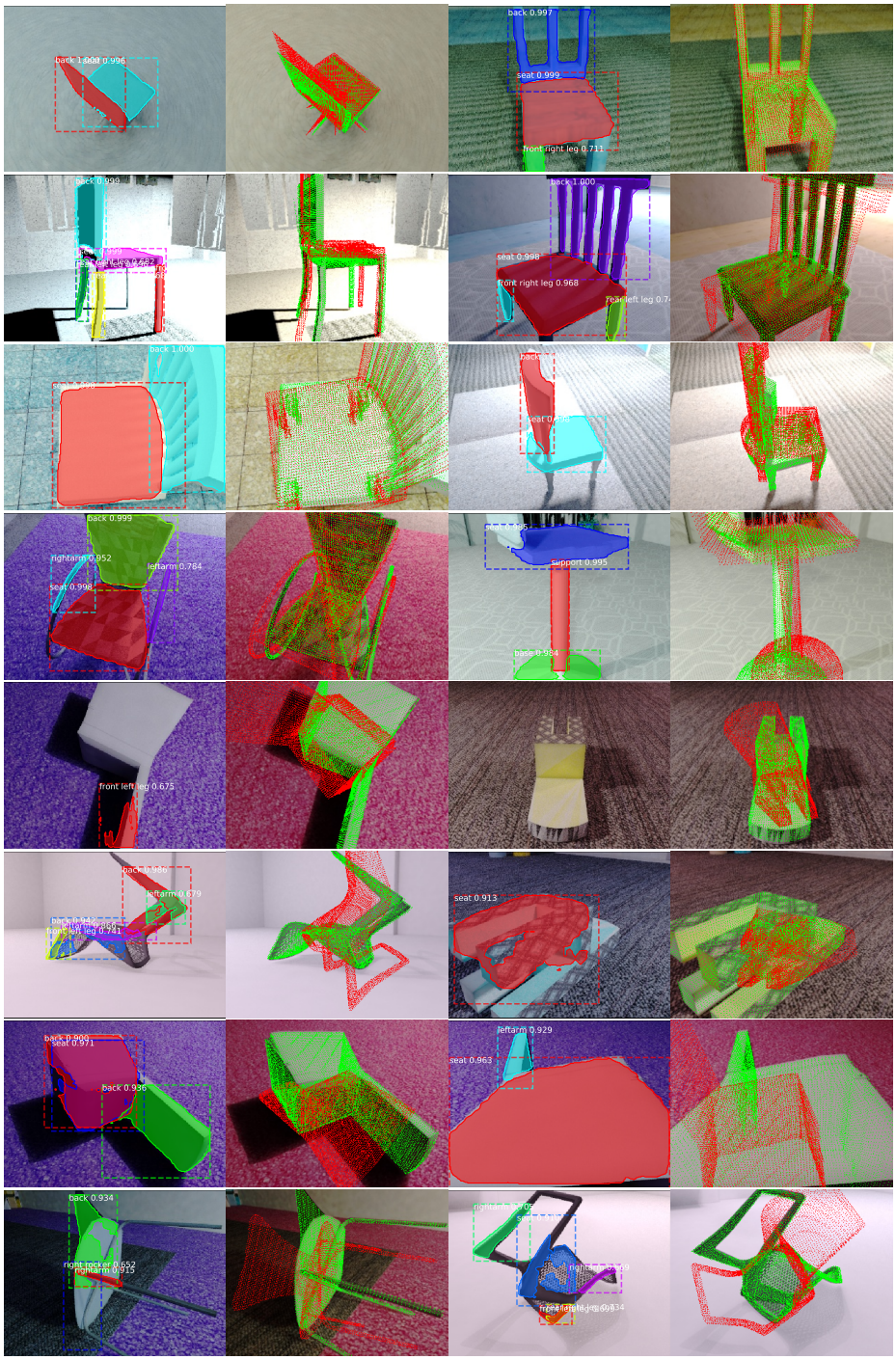


Figure 8.10 Good (top 4 rows) and bad (bottom 4 rows) examples of part detection (paired with the corresponding pose estimate visualization) for the RGB-D modality and Pose+Parts network configuration.

	RGB-D-N @ 0.5	RGB-D-N @ 0.75	RGB-D @ 0.5	RGB-D @ 0.75	RGB-N @ 0.5	RGB-N @ 0.75	RGB @ 0.5	RGB @ 0.75
Pose+Parts	41.4	26.4	**51.0**	37.8	46.8	26.3	6.8	4.2
Pose	42.1	25.6	**49.8**	35.2	47.9	27.9	7.6	5.3
Pose+Parts+so(3)	41.0	26.2	**48.0**	35.6	45.7	27.2	9.1	4.2

Table 8.2 The mAP for PartNet at two common IOU thresholds. This score only considers regions the network detects as compared to ground truth.

8.5.1 Part Detection

Table 8.2 shows the network configuration accuracy for each of the input modalities and 2 IOU thresholds. Table 8.3 shows the same configurations, but in this case the accuracy is computed based on the entire ground truth (i.e., missing detection are counted against the network).

Overall, the results are a little lower than expected, and in two cases, rather surprising. I expected the results to be somewhat higher overall when comparing to the performance of MRCNN on the COCO [131] dataset. With the same network configuration (ResNet-101), the original MRCNN paper reports an AP_{50} of 58.0 with 80 categories, while PartNet yields 51.0 for 13 categories. This may be relatively commensurate, however, when considering the masks can be rather complex for certain parts, due to the generation algorithm. As an example, consider the fourth column of the first and fourth row in Fig. 8.9. A human annotating images directly would probably have not taken the spaces between the slats into consideration when labeling the back and seat parts for those chairs. However, since the image frame annotations are computed automatically from the annotated 3D model, those details are present. This means pixel-per-pixel, PartNet is trying to learn more complicated masks and consequently gets hits harder on the IOU computation.

	RGB-D-N @ 0.5	RGB-D-N @ 0.75	RGB-D @ 0.5	RGB-D @ 0.75	RGB-N @ 0.5	RGB-N @ 0.75	RGB @ 0.5	RGB @ 0.75
Pose+Parts	24.4	17.0	**30.8**	23.3	22.5	12.9	1.1	0.5
Pose	25.6	17.5	**31.4**	21.9	23.1	13.7	1.7	0.9
Pose+Parts+so(3)	24.4	17.0	**30.0**	22.3	22.8	13.5	2.1	0.9

Table 8.3 The mAP for PartNet at two common IOU thresholds. This score considers *all* ground truth regions present in the test images.

Effects from input modality

Here we find the two surprising and unexpected results. First, I expected the RGB-D-N model to perform the best, simply because it was providing more information. Instead, the RGB-D network, a *subset* of the RGB-D-N network, performed the best consistently across the board. All other things equal (including the learning rate and the number of epochs), this may in part be due to greater complexity in the input, i.e., the extra normal data forces the network to work harder to find the features of interest, meaning at the very least it may take more epochs to reach comparable performance. I suspect that performing a hyperparameter search for a better learning rate, and given more epochs, it may outperform the RGB-D network.

Second, the RGB performance is abysmal. This was not at all expected, considering the original MRCNN performs instance segmentatation on RGB images with no additional input. This requires follow-on investigation, but we provide a hypothesis here. We note that when lowering the detection threshold, some of the common parts such as "back" and "seat" are detected, even if the mask is not as well defined. The low learning rate here may have allowed the network to get into a poorly performing local minima, because while the loss decreased during the earlier epochs, it leveled out and failed to decrease for most of the training period. The synthetic data and the nature of the parts may contribute to the low score as well. Modalities that include depth and/or normals provide more information about the shape of parts and their bound-

aries, whereas the distinction in appearance is not as well defined in the synthetic images. In addition, I suspect the network is learning implicit spatial relationships between the parts, and these are less ambiguous in the depth/normal modalities than in color alone, primarily in the depth direction, which may help distinguish the pose of the chair and where the back is with respect to the seat, and therefore left/right legs and armrests. Since the RGB network rarely detects these smaller parts, this may be less of an issue, but should also be a question for further investigation.

Finally, scores may improve across the board simply with additional training time. For all networks except RGB, the validation loss was still decreasing. It is typical to train a network until the validation loss stops decreasing; however, constraints on available GPU usage limited our training time.

Effects from pose variants

The effects caused by the pose network variations seem insignificant, and this was an expected result. While training the network end-to-end means there are gradients that affect shared portions of the network, this primarily affects the feature backbone, which likely has enough "bandwidth" to support features for pose as well as features for parts. In addition, the gradients from the pose network are completely separated from the instance segmentation head networks, even in the Pose+Parts configurations, since I treat the mask generation as a constant operation in the network. This was deliberate, since we were only investigating the parts $\rightarrow$ pose arc, and I wanted no effect (positive or negative) from pose $\rightarrow$ parts in these comparisons.

8.5.2 Pose Estimation

Figure 8.11 shows the full orientation accuracy results across all variants of the network. Like the part detection results, the overall performance here was not too exciting. However, compared to a Gupta et al. [78], our top-performing networks significantly improve on the accuracy for chairs at $Acc_{\frac{\pi}{12}}$ and $Acc_{\frac{\pi}{6}}$[5]. However, compared

[5]To be fair, their network is trained for multiple classes, and we are currently focusing on single class support.

to the state-of-the-art results from Grabner et al. [74], their accuracy for chairs at $Acc_{\frac{\pi}{6}}$ is 80.0 and ours is only 75.0 (Su et al. [197] report 86.0!) While the performance leaves room for improvement (especially considering our use of extended modalities), I am not overly concerned; estimating pose is very important and we shall improve the performance in future iterations, but the primary goal was to investigate how parts affected the pose estimation performance. We discuss this in the next section.

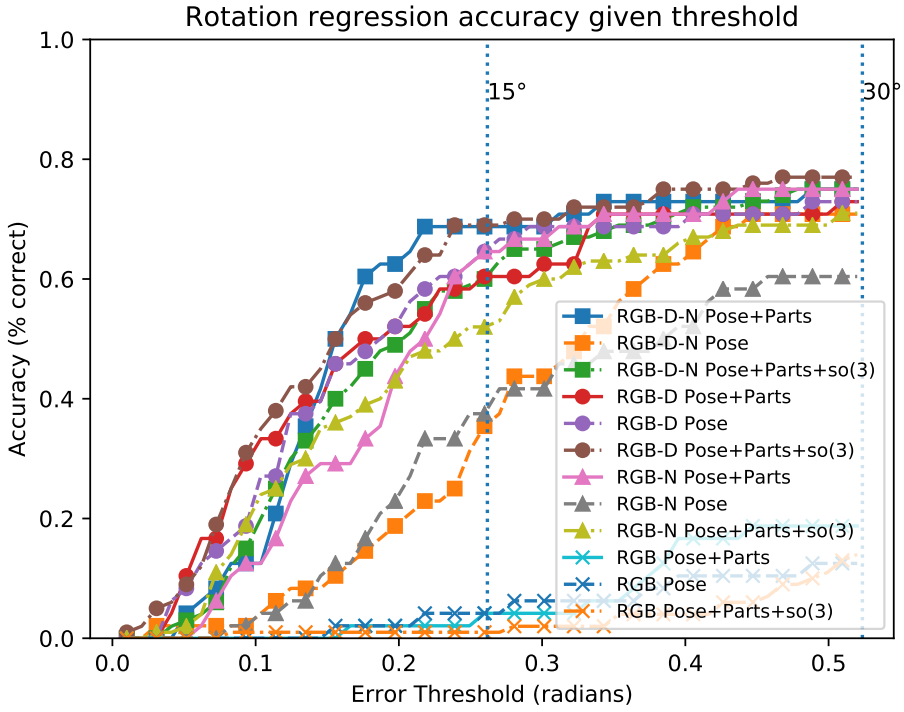


Figure 8.11 Orientation accuracy plotted against an angular threshold. We use the geodesic distance for $SO(3)$ (the minimum angle between orientations) in both the loss function and the metric.

Figure 8.12 shows the full translation accuracy results across all variants of the network. While PartNet translation loss is similar to other implementations, the network implementation in PartNet is simpler than the approach taken in [222]. Their network first estimate votes for x and y centroid offsets for each pixel in the instance mask (i.e., in image coordinates), estimates the depth in the camera frame, and then uses a special voting layer to generate the final regression values. While they report a different ICP-based accuracy metric, the distances they report are under 10 cm.

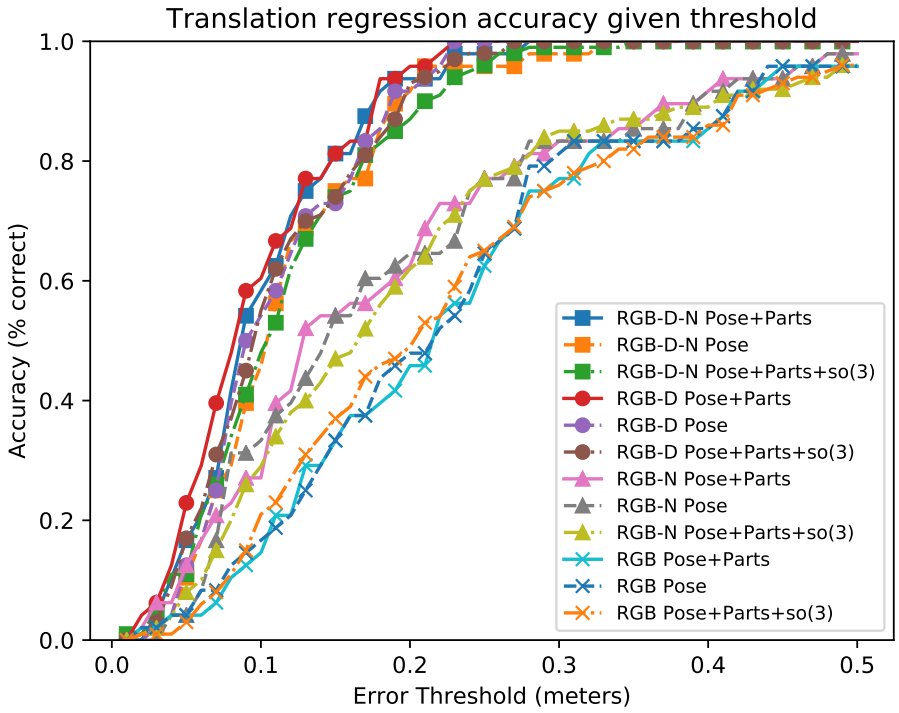


Figure 8.12 Translation accuracy plotted against a distance (in meters) threshold.

Effects from part input

In Fig. 8.13, we show the rotation and translation accuracies broken down by input modality and network configuration. For the RGB-D-N network, this clearly shows how significant the addition of part information is to the orientation accuracy. For the RGB-D networks, the contribution is not *as* significant, but the top network at $Acc_{\frac{\pi}{12}}$ and $Acc_{\frac{\pi}{6}}$ still include parts. The plots also show that the effect on translation estimation accuracy is much less significant.

This supports our hypothesis that providing parts to the pose detection network provides useful information that allows the network to improve the pose estimation performance. There is likely a direct correlation, which we do not explored here, between the accuracy and *completeness* of the part detections and the support to the pose estimation network.

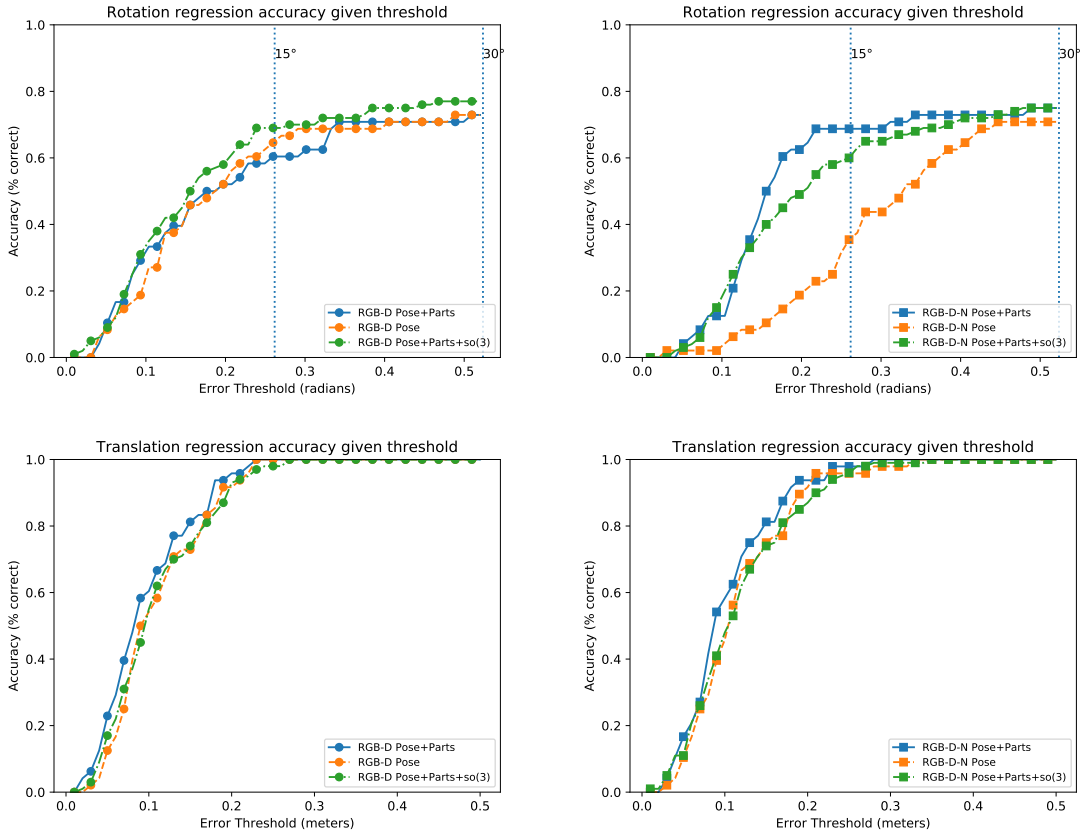


Figure 8.13 Rotation (top row) and translation (bottom row) broken down by input modality: RGB-D (first column) and RGB-D-N (second column).

Effects from orientation encoding

Encoding the orientation estimate as an element of $\mathbf{so}(3)$ as opposed to a unit quaternion has a positive effect on the orientation accuracy for the RGB-D modality, but does not perform as well in the RGB-D-N modality or the RGB-N modality. It's not clear how the addition of the normal information is negatively affecting the performance of the estimation when using $\mathbf{so}(3)$, but the simplest explanation is that it is confounding the features that are specifically useful to the rotation estimation. This is unsatisfactory at best, and warrants additional experiments to uncover an explanation.

8.6 Summary

PartNet represents a step[6] in the direction of perceiving more than just boxes and classes for the objects in the world. The reported experiments show that parts do contribute positively to the pose estimation task, in some cases improving performance at $Acc_{\frac{\pi}{12}}$ by almost 100%. Surprisingly, the RGB-D modality performs the best overall, but further investigation is required to determine whether additional hyperparameter tuning will improve the performance of some of the other modalities.

I believe understanding and recognizing parts is relevant to lifelong learning for semantic representations and ultimately spatial environment understanding. In order to learn incrementally and adapt to new situations and novel objects, I don't believe that we can rely solely on the generalization capacity of a single end-to-end learning algorithm (like deep networks). Instead, I think we really need to focus on more basic and more robust components (e.g., parts), as well as on the recognition of semantic properties. We then send these components and properties to a more flexible reasoning system that can better support learning and adaptation on the fly. In short, we would like a system to be able to recognize new things and learn new objects interactively and experientially, without having to modify or retrain a black box neural network to do so.

Exploring the consequences and challenges of recognizing parts is one step in this journey, since parts may be one way to represent objects at a more basic level. Pose is one of the properties that all objects have, but we really don't want to have to know ahead of time about every single object to be able to say something about its pose. This research shows that detecting semantic parts can be used to improve pose estimation for a known object class. The next steps include expanding this approach to additional objects, looking into improving performance, and investigating unsupervised methods for discovering part classes automatically.

[6] A small step... but a step nevertheless.

Chapter 9

Future Work

In this chapter, I give an overview of where I think this research is heading. In some cases work has already begun, while in other cases the work is only an idea of where I think we can go next.

9.1 Part primitive fitting

One of the many motivating factors while investigating the part network was inducing a network to see simpler subcomponents when looking at an object, i.e., perceiving a decomposition of the object. This was inspired by the work of Tulsiani et al. [203] where they learn to generate volumetric primitives from 3D voxelized objects. While we have demonstrated the feasibility of recognizing and detecting parts in a supervised framework, we want to also show that we can generate cuboid primitives in 3D at the right place. Assuming success in that endeavor, we want to do the same thing without supervision with partial RGB-D views. This is a useful approach, not only because it enables deeper understanding of objects and their sub-parts, relationships and affordances, but because it is a more compact and meaningful representation of an object as compared to just an image, depth map or point cloud. You can reason about parts and simple 3D primitives in a much more effective way. To that end, we have already generated supervised cuboid part primitives that align with the object's point cloud. See Fig. 9.1 for an example set of primitives.

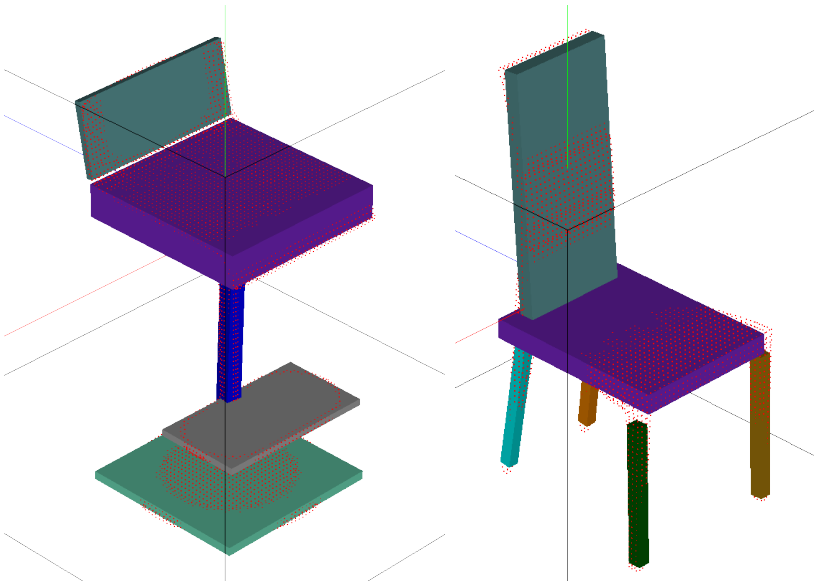


Figure 9.1 Two examples from the primitive ground truth. Note the semantically similar parts are colored (labeled) similarly, and that the chair labeling also supports different chair structures. The primitives are simple cuboids, and are optimized to fit the labeled point cloud (seen as red points) without completely enclosing each part.

9.2 Unsupervised Part Discovery

Given the relatively small set of hand labeled synthetic models, we are interested in combining these supervised data with a larger set of unlabled real data. Specifically, we want to use a combination of supervised labeled data with a bunch of unlabeled data in order to enable unsupervised part *discovery*. In this case, we may be less interested in correct semantic labels and more interested in focusing on decomposing objects into logical components. This seems like a natural extension of our part detection framework, and a possible instantiation of the concept of recognition by parts. This would provide compositional object descriptions and additional understanding that may be used for higher level reasoning and learning tasks.

9.3 Semantic Scene Description

If we start putting everything together, we begin to work towards the goal of fully semantic scene description. This is a superset of the following tasks:

1. object detection (computing pose and relational attributes, not necessarily in-

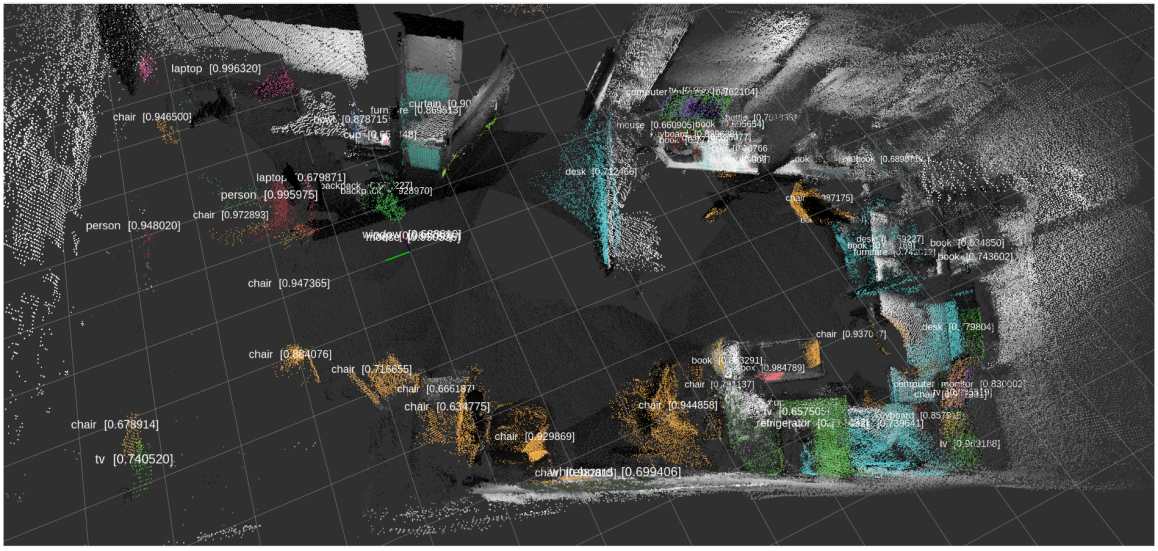


Figure 9.2 This shows a visualization from early conceptual work labeling a map generated using ego-motion estimation and detected object labels back projected onto the points in the map cloud.

stances),

2. object instance detection (computing pose and relational attributes),

3. semantic segmentation (instance labeling, spatial reasoning),

4. and mapping (typically metric).

Semantic scene description aims to provide a rich scene representation based on combining all the components in a manner suitable for reasoning, planning, navigation, and manipulation. The task of meaningfully describing scenes emphasizes object instance detection, pose estimation, spatial relations between instances, and temporal change explanation (i.e., explaining what is different and how it is different). This requires the combination of multiple components and focuses on representations of the environment at both a metric and semantic level. See Fig. 9.2 for an early attempt at this goal.

A semantic scene description must be a usable, scalable and extensible representation. It must combine ego-motion, local environment modeling, and localization over extended periods of time. It must support topological relationships between local metric environment models. It must utilize robust landmarks, account for the

distinction between objects and structure (things and stuff), and support hierarchical organization. The system must support object descriptions that include pose, parts, attributes, and affordances. The representations should be as parsimonious as possible; for example, we cannot simply represent a place as a very dense point clouds; we should not have to use a million points to represent a wall!

Ultimately, the true goal is spatial, temporal, and behavioral knowledge management for intelligent embodied systems that learn from experience.

9.4 Vision for robots with vision

All told, I, personally, am heading towards the far off target of artificial general intelligence and lifelong learning that encapsulates (at a minimum) the capabilities of navigation, place recognition, object recognition and learning. Whether the rest of the robotics and artificial intelligence field agrees with my direction remains to be seen. I've touched on the individual components I believe are required to achieve this goal, albeit with very little detail. What I haven't emphasized enough is the holistic interpretation of how these algorithms work together. In order to achieve the synergy I'm looking for the solutions must be more unified. While it may be there is *not* a unified theory of robot intelligence, I do think that we need to constantly strive for general mechanisms that are applicable for more tasks than the narrow point solutions we tend to come up with in current work.

Chapter 10

Conclusion

My research goals have been, and always will be, to develop algorithms that support and enable intelligence for robotic systems. In this dissertation, I focus on perception algorithms that raise the level of abstraction in one or more ways to enable better understanding of the environment. I also focus on relationships in space, whether it is between the relative pose of visual sensors mounted on a mobile robot, or point locality for temporally consistent over-segmentation, or camera motion in an environment for ego-motion estimation and mapping. Finally, object parts and their relationships provide an increased level of abstraction when recognizing objects and computing information about their pose.

The research in this dissertation addresses several independent aspects of spatial environment perception, and while, separately, these aspects provide support to the operation of embodied intelligent systems, there are several meta-observations I provide as a conclusion. However, I want to first emphasize the very pragmatic approach I take by aiming my work at running on real robots. To a fault, I am constantly concerned with the question of how I can run something on our robot to improve its behavior and performance in the real world. This very practical approach stems from my work as a computer scientist at the CCDC[1] Army Research Laboratory, where our goal is to research and develop new algorithms that will enable future intelligent systems to save the lives of Soldiers.

[1]CCDC: Combat Capabilities Development Command, part of the new Army Futures Command.

Calibration. Calibrating sensors may be an incredibly boring and mundane task. However, I and my colleagues have discovered over and over again that the performance of any algorithm you are trying to run on your system is directly correlated with how well your sensors and platform components are calibrated. How accurate is the pose of the camera known with respect to the laser sensor and how well are both sensors' poses known with respect to a manipulator's end effector? For an intelligent system in real life, a lack of calibration could mean the difference between successfully grasping an object such as a IED versus missing it in just the wrong way. This is not acceptable when time is of the essence and lives are on the line.

Here is another practical fact: whether you have a well calibrated system at the beginning or not, it *will* become uncalibrated over time. MSG-Cal does not yet address this situation. We are actively working on extending the algorithm to support sensors on dynamic joints as well as online calibration maintenance. We have found that our calibration algorithm has performed well for the collection of sensors on our own robots, and others have made use of our system as well. However, there are a few limitations that we will address in future research, including the absence of a more direct camera to camera pairwise calibration routine that could make use of reprojection error and allow for the automatic calibration of stereo sensors.

Motion and environment representation. The ego-motion estimation and high resolution environment mapping work has yielded results that still compare favorably to dense mapping approaches produced more recently, although newer techniques are more efficient. Over the time I have spent focusing on recovering accurate camera motion and generating high-resolution surfel models, I have also realized that we need to allow for some flexibility and inaccuracies if we are going to scale the algorithm to support life-long localization and navigation: the ability to move around in the world and navigate to both well-known and novel locations while improving performance through experience. This will require a combination of accurate localization in local environments (supported by the work in this disseration) as well as robust navigation between local environments through flexible topological relationships and stable landmarks. In addition, it will require the ability to store and retrieve infor-

mation about places at a level much higher than that of surfels. This brings me to the next section on objects...

Object parts, pose, and learning. Objects are a fact of life; we can't live without 'em. That is why so many researchers in computer vision and robotic perception work hard to discover new ways of processing sensor data to produce information about the objects in the world. There will not be intelligent embodied agents that are truly useful to humans until they understand more about objects than they do now. Generating instance segmentations and class labels at 100% accuracy will only be a portion of the story. Parts are yet another "part" of the story. I can not tell you now when the story will end, but I know it will include these things and more. I do believe that it must involve allowing embodied systems to really experience the environment, akin to how human babies develop visual capability. Our brains provide the substrate, but our experience does the programming. Do the robots need 3D sensors? No, but it will probably make it easier. Do the robots need to see in 3D? Absolutely. By seeing in 3D, I mean taking *whatever* the sensors are providing and *understanding* it in a 3 dimensional context, the shapes and spatial relationships intact.

I'm looking forward to continuing this research with you.

—Jason Owens

> You could leave life right now. Let that determine what you do and say and think.
>
> *Marcus Aurelius*

Acronyms

2D two-dimensional. 5, 15, 47, 48, 49, 99

3D three-dimensional. 5, 6, 15, 16, 17, 18, 21, 22, 47, 95, 97

CAD computer aided design. 45, 179

CNN convolutional neural network. 30, 31, 32, 35, 36, 44, 45

CVPR Conference on Computer Vision and Pattern Recognition. 18

DoF degree of freedom. 93

DSO Direct Sparse Odometry. 18

DVO dense visual odometry. xi, 16, 17, 95, 104, 105, 106, 108, 110, 112, 114, 125

EGI extended Gaussian image. 114, 115, 114

FPN feature pyramid network. 171, 181

GICP generalized ICP. 116

GPU graphics processing unit. 20, 21, 22, 23, 108, 109, 110, 111

ICP iterative closest point. 20, 31, 32, 91, 94, 95, 97, 98, 101, 103, 104, 106, 108, 110, 112, 114, 116, 119, 171

IMU inertial measurement unit. 18, 52

INS inertial navigation system. 12

IOU intersection over union. 183, 185

IR infrared. 17

LIDAR light detection and ranging. 23, 48

MRCNN Mask R-CNN. 168, 170, 182, 183, 185, 186

MSG-Cal multi-sensor graph calibration. 67, 68

ORB Oriented Rotated BRIEF. 14, 20

PCL Point Cloud Library. 99

RANSAC random sampling consensus. 68, 72, 90, 93

ReLU rectified linear unit. 171

RGB red, green, and blue. 20, 30, 97

RGB-D red, green, blue, and depth. 15, 16, 17, 16, 17, 18, 20, 21, 30, 89, 95, 110, 115, 132

ROI region of interest. 170, 172

ROS Robot Operating System. 17, 89

SIFT Scale Invariant Feature Transform. 14, 22

SIMT single instruction multiple thread. 109

SLAM simultaneous localization and mapping. 4, 19, 20, 25, 26, 133, 134

SVO Semi-direct Visual Odometry. 18

TSDF truncated signed distance function. 21, 26, 29, 110, 130, 142

VO visual odometry. 16, 18, 19, 20, 26, 103, 114, 118, 119, 121

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012, 00241, ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2012.120.

[2] J. Aleotti and S. Caselli, "A 3D shape segmentation approach for robot grasping by parts," *Robotics and Autonomous Systems*, vol. 60, no. 3, pp. 358–366, Mar. 2012, 00006 bibtex: Aleotti2012, ISSN: 09218890. DOI: 10.1016/j.robot.2011.07.022.

[3] A. Andreopoulos and J. K. Tsotsos, "50 years of object recognition: Directions forward," *Computer Vision and Image Understanding*, vol. 117, pp. 827–891, 8 Aug. 2013, 00001, ISSN: 10773142. DOI: 10.1016/j.cviu.2013.04.005.

[4] Anonymous, "Three Mechanisms of Weight Decay Regularization," in *Submitted to International Conference on Learning Representations*, under review, 2019.

[5] S. Baker and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, Feb. 2004, ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000011205.11775.fd.

[6] V. Balntas, A. Doumanoglou, C. Sahin, J. Sock, R. Kouskouridas, and T.-K. Kim, "Pose Guided RGBD Feature Learning for 3D Object Pose Estimation," IEEE, Oct. 2017, pp. 3876–3884, ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.416.

[7] E. Barnea and O. Ben-Shahar, "Depth Based Object Detection from Partial Pose Estimation of Symmetric Objects," in *Computer Vision ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8693, Cham: Springer International Publishing, 2014, pp. 377–390, ISBN: 978-3-319-10601-4 978-3-319-10602-1. DOI: 10.1007/978-3-319-10602-1_25.

[8] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Computer Vision ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., vol. 3951, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417, ISBN: 978-3-540-33832-1 978-3-540-33833-8. DOI: 10.1007/11744023_32.

[9] C. Beall and F. Dellaert, "Appearance-based localization across seasons in a metric map," *6th PPNIV, Chicago, USA*, 2014.

[10] J. Behley and C. Stachniss, "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments," in *Robotics: SCIENCE and Systems XIV*, Robotics: Science and Systems Foundation, Jun. 26, 2018, ISBN: 978-0-9923747-4-7. DOI: 10.15607/RSS.2018.XIV.016.

[11] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, pp. 509–517, 9 Sep. 1975, ISSN: 00010782. DOI: 10.1145/361002.361007.

[12] P. Bergmann, R. Wang, and D. Cremers, "Online Photometric Calibration for Auto Exposure Video for Realtime Visual Odometry and SLAM," Oct. 5, 2017. arXiv: 1710.02081 [cs].

[13] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, Feb. 1992.

[14] I. Biederman, "Recognition-by-components: A theory of human image understanding.," *Psychological review*, vol. 94, no. 2, p. 115, 1987.

[15] J. Biswas and M. M. Veloso, "Localization and navigation of the cobots over long-term deployments," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1679–1694, 2013.

[16] S. Borra and S. Sarkar, "A framework for performance characterization of intermediate-level grouping modules," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 11, pp. 1306–1312, 1997, 00069.

[17] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6D Object Pose Estimation Using 3D Object Coordinates," in *Computer Vision ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8690, Cham: Springer International Publishing, 2014, pp. 536–551, ISBN: 978-3-319-10604-5 978-3-319-10605-2. DOI: 10.1007/978-3-319-10605-2_35.

[18] G. R. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 1. ed., [Nachdr.], ser. Software That Sees. Beijing: O'Reilly, 2011, 555 pp., OCLC: 838472784, ISBN: 978-0-596-51613-0.

[19] D. Brown, "The simultaneous determination or the orientation and lens distortion of a photogrammetric camera," Air Force Missile Test Center, Patrick AFB, Florida, Technical Report 56-20, 1956.

[20] ——, "Decentering distortion of lenses," *Photogrammetric Engineering*, vol. 32, no. 3, pp. 444–462, May 1966.

[21] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016, ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2016.2624754.

[22] R. Casati and A. C. Varzi, *Parts and Places: THE STRUCTURES of Spatial Representation.* Cambridge: MIT Press, 2003, OCLC: 961856340, ISBN: 978-0-262-27001-4.

[23] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Dec. 9, 2015. arXiv: 1512.03012 [cs].

[24] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the Devil in the Details: Delving Deep into Convolutional Nets," in *Proceedings of the British Machine Vision Conference 2014*, Nottingham: British Machine Vision Association, 2014, pp. 6.1–6.12, ISBN: 978-1-901725-52-0. DOI: 10.5244/C.28.6.

[25] A. Y. C. Chen and J. J. Corso, "Propagating multi-class pixel labels throughout video frames," in *Proceedings of Western New York Image Processing Workshop*, bibtex: Chen2010, 2010.

[26] L.-C. Chen, A. Hermans, G. Papandreou, F. Schroff, P. Wang, and H. Adam, "MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features," Dec. 13, 2017. arXiv: 1712.04837 [cs].

[27] Y. Chen and G. Medioni, "Object modeling by reigstration of multiple range images," in *IEEE INTERNATIONAL CONFERENCE on Robotics and Automation*, Sacramento, California, Apr. 1991, pp. 2724–2729.

[28] C. Choi and H. I. Christensen, "RGB-D object pose estimation in unstructured environments," *Robotics and Autonomous Systems*, vol. 75, pp. 595–613, Jan. 2016, ISSN: 09218890. DOI: 10.1016/j.robot.2015.09.020.

[29] F. Chollet *et al.*, "Keras," 2018.

[30] E. Chown and B. Boots, "Learning cognitive maps: Finding useful structure," in *Robotics and Cognitive Approaches to Spatial Mapping*, M. E. Jeffries and W.-K. Yeap, Eds., ser. Springer Tracts in Advanced Robotics. 2008, pp. 215–236.

[31] S. Christoph Stein, M. Schoeler, J. Papon, and F. Worgotter, "Object partitioning using local convexity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 304–311.

[32] W. Churchill and P. Newman, "Experience-based navigation for long-term localisation," en, *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1645–1661, Dec. 2013, ISSN: 0278-3649. DOI: 10.1177/0278364913499193.

[33] A. E. Conrady, "Decentred lens-systems," *Monthly notices of the Royal Astronomical Society*, vol. 19, pp. 384–390, 1919.

[34] F. R. Corrêa and J. Okamoto, "Semantic mapping with image segmentation using conditional random fields," in *Advanced Robotics, 2009. ICAR 2009. International Conference on*, bibtex: Correa2009, 2009, pp. 1–6.

[35] T. Cour and J. Shi, "Recognizing objects by piecing together the Segmentation Puzzle," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, Minneapolis, MN, USA: IEEE, Jun. 2007, pp. 1–8, ISBN: 978-1-4244-1179-5 978-1-4244-1180-1. DOI: 10.1109/CVPR.2007.383051.

[36] 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*, 1996, pp. 303–312.

[37] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. NieSSner, "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes," Feb. 14, 2017. arXiv: 1702.04405 [cs].

[38] A. Dai, M. Niener, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration," *ArXiv preprint arXiv:1604.01093*, 2016.

[39] (2014). Darpa robotic challenge, [Online]. Available: http://www.theroboticschallenge.org/.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 2009, pp. 248–255.

[41] T.-T. Do, M. Cai, T. Pham, and I. Reid, "Deep-6DPose: Recovering 6D Object Pose from a Single RGB Image," Feb. 28, 2018. arXiv: 1802.10367 [cs].

[42] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3D object recognition," IEEE, Jun. 2010, pp. 998–1005, ISBN: 978-1-4244-6984-0. DOI: 10.1109/CVPR.2010.5540108.

[43] O. Dunkley, J. Engel, J. Sturm, and D. Cremers, "Visual-inertial navigation for a camera-equipped 25g nano-quadrotor," in *IROS2014 aerial open source robotics workshop*, 2014, p. 2.

[44] D. W. Eggert, A. Lorusso, and R. B. Fisher, "Estimating 3-d rigid body transformations: A comparison of four major algorithms," *Mach. Vision Appl.*, vol. 9, no. 5-6, pp. 272–290, Mar. 1997, bibtex: Eggert1997, ISSN: 0932-8092. DOI: 10.1007/s001380050048.

[45] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, Cited by 0058, 2012, pp. 1691–1696.

[46] J. Engel, V. Koltun, and D. Cremers, "Direct sparse Odometry," *ArXiv:1607.02565 [cs]*, Jul. 2016, arXiv: 1607.02565.

[47] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct Monocular slam," in *Computer VisionECCV 2014*, Springer, 2014, pp. 834–849.

[48] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct slam with stereo cameras," in *Proceedings of the 2015 IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 1935–1942.

[49] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual Odometry for a Monocular camera," in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2013.

[50] J. Engel, V. Usenko, and D. Cremers, "A Photometrically Calibrated Benchmark For Monocular Visual Odometry," *ArXiv preprint arXiv:1607.02555*, 2016.

[51] C. Erdogan, M. Paluri, and F. Dellaert, "Planar segmentation of RGBD images using fast linear fitting and markov chain monte carlo," in *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, bibtex: Erdogan2012, 2012, pp. 32–39.

[52] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, "Cascade object detection with deformable part models," *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, 2010.

[53] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008, pp. 1–8.

[54] P. F. Felzenszwalb, "Representation and detection of deformable shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 2, pp. 208–220, 2005.

[55] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, no. 2, pp. 167–181, Sep. 2004, bibtex: Felzenszwalb2004, ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000022288.19776.77.

[56] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *International Journal of Computer Vision*, vol. 61, pp. 55–79, 1 2005, 01197, ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000042934.15159.49.

[57] Fidler, Dickinson, and Urtasun, "3d object detection and viewpoint estimation with deformable 3d cuboid model," presented at the NIPS, 2012.

[58] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, pp. 1–9, Mar. 1, 1974, ISSN: 1432-0525. DOI: 10.1007/BF00288933.

[59] R. Finman, T. Whelan, M. Kaess, and J. J. Leonard, "Efficient incremental map segmentation in dense RGB-D maps," in *International Conference on Robotics and Automation*, 00000, 2014.

[60] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "FlowNet: Learning Optical Flow with Convolutional Networks," *ArXiv preprint arXiv:1504.06852*, 2015, 00000.

[61] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, 1981.

[62] M. A. Fischler and R. A. Elschlager, "The Representation and Matching of Pictorial Structures," *IEEE Transactions on Computers*, vol. C-22, no. 1, pp. 67–92, Jan. 1973, ISSN: 0018-9340. DOI: 10.1109/T-C.1973.223602.

[63] M. a. Fischler and R. a. Elschlager, "The representation and matching of pictorial structures," *IEEE Transactions on Computers*, vol. C-22, pp. 67–92, 1 1973, ISSN: 0018-9340. DOI: 10.1109/T-C.1973.223602.

[64] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 15–22. DOI: 10.1109/ICRA.2014.6906584.

[65] K. Fragkiadaki, G. Zhang, and J. Shi, "Video segmentation by tracing discontinuities in a trajectory embedding," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 00012, IEEE, 2012, pp. 1846–1853.

[66] X. Fu, F. Zhu, Q. Wu, Y. Sun, R. Lu, and R. Yang, "Real-Time Large-Scale Dense Mapping with Surfels," *Sensors (Basel, Switzerland)*, vol. 18, no. 5, May 9, 2018, ISSN: 1424-8220. DOI: 10.3390/s18051493. pmid: 29747450.

[67] B. Fulkerson, A. Vedaldi, and S. Soatto, "Class segmentation and object localization with superpixel neighborhoods," in *Computer Vision, 2009 IEEE 12th International Conference on*, bibtex: Fulkerson2009, 2009, pp. 670–677.

[68] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: ELEMENTS of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN: 0-201-63361-2.

[69] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[70] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, 2011, pp. 963–968.

[71] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy, "Geometrically stable sampling for the ICP algorithm," IEEE, 2003, pp. 260–267, ISBN: 978-0-7695-1991-3. DOI: 10.1109/IM.2003.1240258.

[72] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *ARXIV preprint arXiv:1311.2524*, 2013, 00005 bibtex: Girshick2013.

[73] R. Girshick, F. Iandola, T. Darrell, and J. Malik, "Deformable part models are convolutional neural networks," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 437–446, ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298641.

[74] A. Grabner, P. M. Roth, and V. Lepetit, "3D Pose Estimation and 3D Model Retrieval for Objects in the Wild," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[75] K. Grauman, "See, Hear, Move: Towards Embodied Visual Perception," 2018.

[76] M. Grundmann, V. Kwatra, M. Han, and I. Essa, "Efficient hierarchical graph-based video segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 00144, IEEE, 2010, pp. 2141–2148.

[77] S. Gupta, P. Arbelaez, R. Girshick, and J. Malik, "Aligning 3D models to RGB-D images of cluttered scenes," IEEE, Jun. 2015, pp. 4731–4740, ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7299105.

[78] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik, "Inferring 3D Object Pose in RGB-D Images," Feb. 16, 2015. arXiv: 1502.04652 [cs].

[79] S. Gupta, P. Arbelaez, and J. Malik, "Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images," IEEE, Jun. 2013, pp. 564–571, ISBN: 978-0-7695-4989-7. DOI: 10.1109/CVPR.2013.79.

[80] S. Harnad, "The Symbol Grounding Problem," *Physica D*, vol. 42, pp. 335–346, 1990.

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

[82] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," Mar. 20, 2017. arXiv: 1703.06870 [cs].

[83] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *ArXiv preprint arXiv:1512.03385*, 2015.

[84] D. M. Healy, D. N. Rockmore, P. J. Kostelec, and S. Moore, "FFTs for the 2-Sphere-Improvements and Variations," *Journal of Fourier Analysis and Applications*, vol. 9, no. 4, pp. 341–385, 2003.

[85] G. Heitz and D. Koller, "Learning spatial context: Using stuff to find things," *Computer VisionECCV 2008*, pp. 30–43, 2008.

[86] P. Henry, D. Fox, A. Bhowmik, and R. Mongia, "Patch volumes: Segmentation-based consistent mapping with RGB-D cameras," 00002 bibtex: Henry2013, IEEE, Jun. 2013, pp. 398–405, ISBN: 978-0-7695-5067-1. DOI: 10.1109/3DV.2013.59.

[87] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, Apr. 2012, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364911434148.

[88] R. Herzig, E. Levi, H. Xu, E. Brosh, A. Globerson, and T. Darrell, "Classifying Collisions with Spatio-Temporal Action Graph Networks," Dec. 4, 2018. arXiv: 1812.01233 [cs].

[89] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit, "Gradient response maps for real-time detection of texture-less objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[90] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," *Asian Conference on Computer Vision*, 2012.

[91] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab, "Dominant orientation templates for real-time detection of texture-less objects," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

[92] S. Hinterstoisser, V. Lepetit, N. Rajkumar, and K. Konolige, "Going Further with Point Pair Features," vol. 9907, pp. 834–848, 2016. DOI: 10.1007/978-3-319-46487-9_51. arXiv: 1711.04061 [cs].

[93] hokuyo. (2018). Scanning Rangefinder Distance Data Output/UTM-30LX Product Details | HOKUYO AUTOMATIC CO., LTD., [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=169.

[94] D. Holz and S. Behnke, "Approximate triangulation and region growing for efficient segmentation and smoothing of range images," en, *Robotics and Autonomous Systems*, Apr. 2014, 00000, ISSN: 09218890. DOI: 10.1016/j.robot.2014.03.013.

[95] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," presented at the ACM SIGGRAPH, 1992, pp. 71–78.

[96] B. K. P. Horn, "Extended Gaussian images," *Proceedings of the IEEE*, vol. 72, no. 12, pp. 1671–1686, Dec. 1984, ISSN: 0018-9219. DOI: 10.1109/PROC.1984.13073.

[97] H. Hu, D. Munoz, J. Bagnell, and M. Hebert, "Efficient 3-d scene analysis from streaming data," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 00009, May 2013, pp. 2297–2304. DOI: 10.1109/ICRA.2013.6630888.

[98] 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 *3D Vision (3DV), 2016 Fourth International Conference on*, IEEE, 2016, pp. 92–101.

[99] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 559–568.

[100] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM, 2011, pp. 559–568.

[101] F. Jäkel, M. Singh, F. A. Wichmann, and M. H. Herzog, "An overview of quantitative approaches in Gestalt perception," en, *Vision Research*, vol. 126, pp. 3–8, Sep. 2016, ISSN: 00426989. DOI: 10.1016/j.visres.2016.06.004.

[102] M. E. Jefferies and W. K. Yeap, Eds., *Robotics and Cognitive Approaches to Spatial Mapping*, ser. Springer Tracts in Advanced Robotics v. 38, OCLC: ocn173721135, Berlin ; New York: Springer, 2008, 328 pp., ISBN: 978-3-540-75386-5.

[103] H. Johannsson, "Toward lifelong visual localization and mapping," eng, Thesis, Massachusetts Institute of Technology, 2013.

[104] W. Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 34, no. 5, pp. 827–828, 1978.

[105] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, Feb. 1, 2012, ISSN: 0278-3649. DOI: 10.1177/0278364911430419.

[106] P. KaewTraKulPong and R. Bowden, "An improved adaptive background mixture model for real-time tracking with shadow detection," in *Proc. 2nd European Workshop on Advanced Video Based Surveillance Systems*, vol. 25, 2001, pp. 1–5.

[107] C. Kerl, J. Sturm, and D. Cremers, "Robust odometry estimation for RGB-D cameras," in *International Conference on Robotics and Automation*, bibtex: Kerl, 2013.

[108] K. Koffka, *Principles of gestalt psychology*, eng. New York: Harcourt, Brace, & World, 1935, OCLC: 248396251.

[109] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien, "Spectral surface reconstruction from noisy point clouds," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 00157, 2004, pp. 11–21.

[110] K. Konolige and J. Bowman, "Towards lifelong visual maps," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, IEEE, 2009, pp. 1156–1163.

[111] G. Kootstra, N. Bergstrom, and D. Kragic, "Using symmetry to select fixation points for segmentation," in *Pattern Recognition (ICPR), 2010 20th International Conference on*, IEEE, 2010, pp. 3894–3897.

[112] G. Kootstra, N. Bergström, and D. Kragic, "Fast and automatic detection and segmentation of unknown objects," in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2010, pp. 442–447.

[113] ——, "Gestalt principles for attention and segmentation in natural and artificial vision systems," in *ICRA 2011 Workshop on Semantic Perception, Mapping and Exploration (SPME), Shanghai, China*, eSMCs, 2011.

[114] G. Kootstra and D. Kragic, "Fast and bottom-up object detection, segmentation, and evaluation using Gestalt principles," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 3423–3428.

[115] P. J. Kostelec and D. N. Rockmore, "FFTs on the Rotation Group," Santa Fe Institute's Working Paper Series, 03-11-060, 2003.

[116] T. Krajník, J. P. Fentanes, M. Hanheide, and T. Duckett, "Persistent localization and life-long mapping in changing environments using the frequency map enhancement," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 4558–4563. DOI: 10.1109/IROS.2016.7759671.

[117] A. Krull, E. Brachmann, F. Michel, M. Y. Yang, S. Gumhold, and C. Rother, "Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images," Aug. 19, 2015. arXiv: 1508.04546 [cs].

[118] A. Krull, E. Brachmann, S. Nowozin, F. Michel, J. Shotton, and C. Rother, "PoseAgent: Budget-Constrained 6D Object Pose Estimation via Reinforcement Learning," IEEE, Jul. 2017, pp. 2566–2574, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.275.

[119] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," English, in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2011, pp. 3607–3613, ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5979949.

[120] M. Labbe and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, pp. 734–745, 3 Jun. 2013, 00001, ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2013.2242375.

[121] Q. V. Le and A. Y. Ng, "Joint calibration of multiple sensors," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, IEEE, 2009, pp. 3651–3658.

[122] S.-O. Lee, H. Lim, H.-G. Kim, and S. C. Ahn, "RGB-D fusion: Real-time robust tracking and dense mapping with RGB-D data fusion," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 00000, IEEE, 2014, pp. 2749–2754.

[123] V. Lempitsky, A. Vedaldi, and A. Zisserman, "A pylon model for semantic segmentation," 2011, bibtex: Lempitsky2011.

[124] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi, "Turbopixels: Fast superpixels using geometric flows," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 12, pp. 2290–2297, 2009, 00241.

[125] J. Levinson and S. Thrun, "Automatic online calibration of cameras and lasers," in *Robotics: Science and Systems*, 2013.

[126] C. Li, J. Bai, and G. D. Hager, "A Unified Framework for Multi-View Multi-Class Object Pose Estimation," Mar. 21, 2018. arXiv: 1803.08103 [cs].

[127] H. Li and N. Fawzi, "Comprehensive extrinsic calibration of a camera and a 2d laser scanner for a ground vehicle," no. RT-0438, p. 24, Jul. 2013.

[128] S. Li, A. Handa, Y. Zhang, and A. Calway, "Hdrfusion: HDR slam using a low-cost auto-exposure RGB-D sensor," *ArXiv preprint arXiv:1604.00895*, 2016.

[129] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep Iterative Matching for 6D Pose Estimation," Mar. 31, 2018. arXiv: 1804.00175 [cs].

[130] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," Dec. 9, 2016. arXiv: 1612.03144 [cs].

[131] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common objects in context," *ArXiv:1405.0312 [cs]*, May 2014, arXiv: 1405.0312.

[132] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[133] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision.," *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

[134] W. Maddern, A. Harrison, and P. Newman, "Lost in translation (and rotation): Rapid extrinsic calibration for 2d and 3d LIDARs," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, 2012, pp. 3096–3102.

[135] S. Mahendran, H. Ali, and R. Vidal, "3D Pose Regression using Convolutional Neural Networks," Aug. 18, 2017. arXiv: 1708.05628 [cs].

[136] A. Makadia and K. Daniilidis, "Rotation recovery from spherical images without correspondences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1170–1175, 2006.

[137] D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. New York, NY, USA: Henry Holt and Co., Inc., 1982, ISBN: 0716715678.

[138] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, 02110, vol. 2, IEEE, 2001, pp. 416–423.

[139] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison, "SceneNet RGB-D: 5m Photorealistic images of synthetic indoor trajectories with ground truth," *ArXiv preprint arXiv:1612.05079*, 2016.

[140] D. Meagher, "Octree encoding: A new technique for the representation, manipulation, and display of arbitrary 3-D objects by computer," Rensselaer Polytechnic Institute, Image Processing Laboratory, Technical Report IPL-TR-80-111, Oct. 1980, p. 121.

[141] F. Michel, A. Kirillov, E. Brachmann, A. Krull, S. Gumhold, B. Savchynskyy, and C. Rother, "Global Hypothesis Generation for 6D Object Pose Estimation," IEEE, Jul. 2017, pp. 115–124, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.20.

[142] S. Miller, A. Teichman, and S. Thrun, "Unsupervised extrinsic calibration of depth sensors in dynamic scenes," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[143] F. M. Mirzaei, D. G. Kottas, and S. I. Roumeliotis, "3d LIDAR-camera intrinsic and extrinsic calibration: Identifiability and analytical least-squares-based initialization.," *I. J. Robotic Res.*, vol. 31, no. 4, pp. 452–467, 2012.

[144] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding," p. 10, Dec. 6, 2018.

[145] G. Mori, "Guiding model search using segmentation," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, 00179, vol. 2, IEEE, 2005, pp. 1417–1423.

[146] P. Mühlfellner, "Lifelong visual localization for automated vehicles," eng, PhD, Halmstad University, Sweden, 2015.

[147] A. C. Müller and S. Behnke, "Learning depth-sensitive conditional random fields for semantic segmentation of RGB-D images," in *International Conference on Robotics and Automation*, 00000, 2014.

[148] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2015.2463671.

[149] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source slam system for monocular, stereo and RGB-D cameras," *ArXiv preprint arXiv:1610.06475*, 2016.

[150] V. Narayanan and M. Likhachev, "Deliberative object pose estimation in clutter," IEEE, May 2017, pp. 3125–3130, ISBN: 978-1-5090-4633-1. DOI: 10.1109/ICRA.2017.7989357.

[151] R. Nevatia and T. O. Binford, "Description and recognition of curved objects," *Artificial Intelligence*, vol. 8, no. 1, pp. 77–98, Feb. 1, 1977, ISSN: 0004-3702. DOI: 10.1016/0004-3702(77)90006-6.

[152] 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 *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, 2011, pp. 127–136.

[153] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 343–352.

[154] P. Núñez, P. Drews Jr, R. Rocha, and J. Dias, "Data fusion calibration for a 3d laser range finder and a camera using inertial data.," in *ECMR*, 2009, pp. 31–36.

[155] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 00031, 2011, pp. 3400–3407.

[156] P. R. Osteen, J. L. Owens, and C. C. Kessens, "Online egomotion estimation of RGB-D sensors using spherical harmonics," in *Robotics and Automation (ICRA), 2012 IEEE International Conference On*, IEEE, 2012, pp. 1679–1684.

[157] J. L. Owens, P. R. Osteen, and K. Daniilidis, "Temporally consistent segmentation of point clouds," in *SPIE DEFENSE+ Security*, International Society for Optics and Photonics, 2014, 90840H–90840H.

[158] ——, "MSG-Cal: Multi-sensor Graph-based Calibration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[159] J. Owens and M. Fields, "Incremental region segmentation for hybrid map generation," in *Army Science Conference*, 2010, pp. 281–286.

[160] J. Owens, P. Osteen, and M. Fields, "Autonomous exploration and mapping of unknown environments," in *SPIE DEFENSE, Security, and Sensing*, International Society for Optics and Photonics, 2012, pp. 838 717–838 717.

[161] J. Owens and P. Osteen, "Ego-motion and tracking for continuous object learning: A brief survey," US Army Research Laboratory, Aberdeen, MD, Technical Report ARL-TR-8167, Sep. 2017, p. 50.

[162] C. Papazov, S. Haddadin, S. Parusel, K. Krieger, and D. Burschka, "Rigid 3d geometry matching for grasping of known objects in cluttered scenes," *The International Journal of Robotics Research*, vol. 31, pp. 538–553, 4 2012.

[163] J. Papon, A. Abramov, M. Schoeler, and F. Worgotter, "Voxel cloud connectivity segmentation - supervoxels for point clouds," 00002, IEEE, Jun. 2013, pp. 2027–2034, ISBN: 978-0-7695-4989-7. DOI: 10.1109/CVPR.2013.264.

[164] C. Park, S. Kim, P. Moghadam, C. Fookes, and S. Sridharan, "Probabilistic Surfel Fusion for Dense LiDAR Mapping," Sep. 5, 2017. arXiv: 1709.01265 [cs].

[165] M. Pauly, N. J. Mitra, and L. Guibas, "Uncertainty and variability in point cloud surface data," in *Symposium on point-based graphics*, 00078, vol. 9, 2004.

[166] B. Pepik, P. Gehler, M. Stark, and B. Schiele, "3d$\hat{2}$pm - 3d deformable part models," in *European Conference on Computer Vision (ECCV)*, Firenze, Italy, 2012.

[167] 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*, ser. SIGGRAPH '00, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 335–342, ISBN: 1-58113-208-5. DOI: 10.1145/344779.344936.

[168] F. Pomerleau, F. Colas, and R. Siegwart, "A Review of Point Cloud Registration Algorithms for Mobile Robotics," *Foundations and Trends in Robotics*, vol. 4, no. 1, pp. 1–104, 2015. DOI: 10.1561/2300000035.

[169] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak, "Fast plane detection and polygonalization in noisy 3d range images," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, 2008, pp. 3378–3383.

[170] H. Pottmann, S. Leopoldseder, and M. Hofer, "Registration without ICP," *Computer Vision and Image Understanding*, vol. 95, no. 1, pp. 54–71, 2004. DOI: 10.1016/j.cviu.2004.04.002.

[171] D. Prakel, *The Visual Dictionary of Photography*, ser. AVA Academia. Lausanne: AVA Publ, 2010, 288 pp., OCLC: 934382255, ISBN: 978-2-940411-04-7.

[172] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," Jun. 7, 2017. arXiv: 1706.02413 [cs].

[173] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: An open-source robot operating system," in *International Conference on Robotics and Automation, Workshop on Open-Source Robotics*, 00865, 2009.

[174] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[175] X. Ren and J. Malik, "Learning a classification model for segmentation," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 00595, IEEE, 2003, pp. 10–17.

[176] R. Rios-Cabrera and T. Tuytelaars, "Discriminatively Trained Templates for 3D Object Detection: A Real Time Scalable Approach," in *2013 IEEE International Conference on Computer Vision*, Dec. 2013, pp. 2048–2055. DOI: 10.1109/ICCV.2013.256.

[177] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, pp. 2564–2571.

[178] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," PhD, Technische Universität München, Munich, Nov. 2010, 284 pp.

[179] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[180] ——, "3D is here: Point cloud library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, 00491 bibtex: Rusu_ICRA2011_PCL, May 2011.

[181] D. Scaramuzza and F. Fraundorfer, "Visual Odometry [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80–92, Dec. 2011, ISSN: 1070-9932. DOI: 10.1109/MRA.2011.943233.

[182] D. Scaramuzza, A. Harati, and R. Siegwart, "Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes," English, 2007, ISBN: 8173192219 9788173192210.

[183] N. Schaeffer, "Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations," *Geochemistry, Geophysics, Geosystems*, vol. 14, no. 3, pp. 751–758, Mar. 1, 2013, ISSN: 1525-2027. DOI: 10.1002/ggge.20071.

[184] T. Schöps, J. Engel, and D. Cremers, "Semi-dense visual odometry for Ar on a smartphone," in *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, IEEE, 2014, pp. 145–150.

[185] T. Schöps, T. Sattler, and M. Pollefeys, "SurfelMeshing: Online Surfel-Based Mesh Reconstruction," Oct. 1, 2018. arXiv: 1810.00729 [cs].

[186] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in *Proceedings of Robotics: SCIENCE and Systems*, Seattle, USA, Jun. 2009.

[187] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, pp. 888–905, 8 2000, 07995.

[188] ——, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, 2000, 07995 bibtex: Shi2000.

[189] N. Silberman and R. Fergus, "Indoor scene segmentation using a structured light sensor," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, bibtex: Silberman2011, 2011, pp. 601–608.

[190] S. Song, S. P. Lichtenberg, and J. Xiao, "SUN RGB-D: A RGB-D scene understanding benchmark suite," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 567–576, ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298655.

[191] A. N. Stein, T. S. Stepleton, and M. Hebert, "Towards unsupervised whole-object segmentation: Combining automated matting with boundary detection," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 00026 bibtex: Stein2008, 2008, pp. 1–8.

[192] F. Steinbrucker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense RGB-D images," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, bibtex: Steinbrucker2011, 2011, pp. 719–722.

[193] F. Steinbrücker, J. Sturm, and D. Cremers, "Volumetric 3d mapping in real-time on a cpu," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 2021–2028.

[194] A. R. J. Strom and E. Olson, "Aprilcal: Assisted and repeatable camera calibration,"

[195] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[196] ——, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the IEEE Int. Conf. on Intelligent Robot Systems (IROS)*, 2012.

[197] H. Su, C. R. Qi, Y. Li, and L. Guibas, "Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views," *ArXiv preprint arXiv:1505.05641*, 2015.

[198] J. Tighe and S. Lazebnik, "Superparsing," *International Journal of Computer Vision*, vol. 101, no. 2, pp. 329–349, 2013, bibtex: Tighe2013-1.

[199] G. D. Tipaldi, D. Meyer-Delius, and W. Burgard, "Lifelong localization in changing environments," en, *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1662–1678, Dec. 2013, ISSN: 0278-3649. DOI: 10.1177/0278364913502830.

[200] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Bombay, India: Narosa Publishing House, 1998, pp. 839–846, ISBN: 978-81-7319-221-0. DOI: 10.1109/ICCV.1998.710815.

[201] R. Tron, P. Osteen, J. Owens, and K. Daniilidis, "Pose Optimization for the Registration of Multiple Heterogeneous Views," in *Multi-View Geometry in Robotics at Robotics, Science, and Systems*, 2014.

[202] ——, "Pose optimization for the registration of multiple heterogeneous views," in *Workshop on Multi-View Geometry in Robotics*, 2014.

[203] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik, "Learning Shape Abstractions by Assembling Volumetric Primitives," Dec. 1, 2016. arXiv: `1612.00404 [cs]`.

[204] R. Unnikrishnan and M. Hebert, "Fast extrinsic calibration of a laser rangefinder to a camera," Robotics Institute, Tech. Rep. CMU-RI-TR-05-09, Jul. 2005.

[205] F. Vasconcelos, J. P. Barreto, and U. Nunes, "A minimal solution for the extrinsic calibration of a camera and a laser-rangefinder," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 11, pp. 2097–2107, 2012.

[206] I. Velodyne LiDAR, *Velodyne LiDAR*, Available: http://velodynelidar.com/, Jan. 2017.

[207] *Visual cortex*, en, in *Wikipedia, the free encyclopedia*, Page Version ID: 581032399, Dec. 21, 2013.

[208] R. Wagner, O. Birbach, and U. Frese, "Rapid development of manifold-based graph optimization systems for multi-sensor calibration and SLAM," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, IEEE, 2011, pp. 3305–3312.

[209] F. Wang and Z. Zhao, "A survey of iterative closest point algorithm," in *2017 Chinese Automation Congress (CAC)*, Jinan: IEEE, Oct. 2017, pp. 4395–4399, ISBN: 978-1-5386-3524-7. DOI: `10.1109/CAC.2017.8243553`.

[210] Weikersdorfer, Gossow, and Beetz, "Depth-adaptive superpixels," in *Int'l Conference on Pattern Recognition*, bibtex: Weikersdorfer2012, 2012.

[211] T. Weise, T. Wismer, B. Leibe, and L. Van Gool, "In-hand scanning with online loop closure," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, 2009, pp. 1630–1637.

[212] S. Weiss, *Dealing with scale*, Tutorial, Available: http://frc.ri.cmu.edu/ kaess/vslam_cvpr14/media/VSLAM-Tutorial-CVPR14-A21-DealingWithScale.pdf, CVPR, Jun. 2014.

[213] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust tracking for real-time dense RGB-D mapping with kintinuous," 2012, bibtex: Whelan2012.

[214] 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," en, *The International Journal of Robotics Research*, Dec. 2014, 00000, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364914551008.

[215] ——, "Real-time large-scale dense RGB-D slam with volumetric fusion," en, *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, Apr. 2015, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364914551008.

[216] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "Elasticfusion: Dense slam without a pose graph," *Proc. Robotics: Science and Systems, Rome, Italy*, 2015.

[217] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard, "Kintinuous: Spatially extended KinectFusion," in *Robotics, Science and Systems RGB-D Workshop 2012*, 2012.

[218] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Machine Learning*, 1992, pp. 229–256.

[219] P. Wohlhart and V. Lepetit, "Learning Descriptors for Object Recognition and 3D Pose Estimation," pp. 3109–3118, Jun. 2015. DOI: 10.1109/CVPR.2015.7298930. arXiv: 1502.05908 [cs].

[220] C. K. Wong, J. Schmidt, and W. K. Yeap, "Using a Mobile Robot for Cognitive Mapping.," in *IJCAI*, 2007, pp. 2243–2249.

[221] J. Wu, T. Xue, J. J. Lim, Y. Tian, J. B. Tenenbaum, A. Torralba, and W. T. Freeman, "3D Interpreter Networks for Viewer-Centered Wireframe Modeling," *International Journal of Computer Vision*, vol. 126, no. 9, pp. 1009–1026, Sep. 2018, ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-018-1074-6. arXiv: 1804.00782.

[222] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," Nov. 1, 2017. arXiv: 1711.00199 [cs].

[223] ——, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," in *Robotics: SCIENCE and Systems (RSS)*, 2018.

[224] C. Xu and J. J. Corso, "Evaluation of super-voxel methods for early video processing," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 00033, IEEE, 2012, pp. 1202–1209.

[225] C. Xu, C. Xiong, and J. J. Corso, "Streaming hierarchical video segmentation," in *Computer VisionECCV 2012*, 00021 bibtex: Xu2012, Springer, 2012, pp. 626–639.

[226] S. Yang and C. Wang, "Simultaneous egomotion estimation, segmentation, and moving object detection," *Journal of Field Robotics*, 2011, 00007 bibtex: Yang2011.

[227] W. K. Yeap and M. E. Jefferies, "Computing a representation of the local environment," *Artificial Intelligence*, vol. 107, no. 2, pp. 265–301, Feb. 1999, ISSN: 00043702. DOI: 10.1016/S0004-3702(98)00111-8.

[228] L. Yi, L. Guibas, A. Hertzmann, V. G. Kim, H. Su, and E. Yumer, "Learning Hierarchical Shape Segmentation and Labeling from Online Repositories," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–12, Jul. 20, 2017, ISSN: 07300301. DOI: 10.1145/3072959.3073652. arXiv: 1705.01661.

[229] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6D pose estimation in the Amazon Picking Challenge," IEEE, May 2017, pp. 1386–1383, ISBN: 978-1-5090-4633-1. DOI: 10.1109/ICRA.2017.7989165.

[230] J. Zhang, *Loam_continuous - ROS wiki*, Available: http://wiki.ros.org/loam_continuous, Jan. 2017.

[231] J. Zhang and S. Singh, "Loam: Lidar Odometry and mapping in real-time," in *Robotics: SCIENCE and Systems Conference (RSS 2014)*, 2014.

[232] ——, "Visual-lidar odometry and mapping: Low-drift, robust, and fast," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 2174–2181.

[233] Q. Zhang and R. Pless, "Extrinsic calibration of a camera and laser range finder (improves camera calibration)," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, Sendai, Japan, Oct. 2004, pp. 2301–2306. DOI: 10.1109/IROS.2004.1389752.

[234] Z. Zhang, C. Forster, and D. Scaramuzza, "Active Exposure Control for Robust Visual Odometry in HDR Environments," in *ICRA*, 2017.

[235] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A Modern Library for 3D Data Processing," *ArXiv:1801.09847*, 2018.

[236] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis, "Single image 3d object detection and pose estimation for grasping," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3936–3943. DOI: 10.1109/ICRA.2014.6907430.