

©Copyright 2014

Evan Herbst





# 3-D Object Segmentation from Motion Cues

Evan Herbst

A dissertation submitted in partial fulfillment of the  
requirements for the degree of

Doctor of Philosophy

University of Washington

2014

Reading Committee:

Dieter Fox, Chair

Xiaofeng Ren

Steve Seitz

Thomas Duchamp

Program Authorized to Offer Degree:  
Computer Science and Engineering



University of Washington

**Abstract**

3-D Object Segmentation from Motion Cues

Evan Herbst

Chair of the Supervisory Committee:  
Professor Dieter Fox  
Computer Science and Engineering

In this work we tackle the problem of segmenting rigid objects using various types of motion cues. The setting is that of personal robotics, in which robots generally have cameras and arms, and are capable of manipulating objects at close range. Object segmentation is the problem of determining the boundaries, in a 2-D view or full 3-D, of objects present in a scene. In this work we make use of motion caused both by the robot and by other agents to perform segmentation. Similarly, we can use motion for segmentation whether the actual movement was observed by a robot or not. Our motion estimation algorithms are based on change detection, point feature matching, and scene flow.

We build on recent static online SLAM techniques to perform online segmentation and 3-D modeling of rigid objects. Performing segmentation online allows us to also do active segmentation, in which a robot moves its camera and/or manipulates objects in the scene in order to improve its segmentation and/or geometric object models.

We perform object discovery on the partial object models we obtain from any of these segmentation methods. We design an objective function using shape and color information to be minimized to align partial object models, and we can use information from this alignment process for clustering of object views into instances. This gives us further information on the movement of individual objects over time.

All of this work is performed with RGB-D cameras, which capture both color and depth images. Our methods are designed to work with RGB-D image data and to handle the behavior of RGB-D sensors with respect to missing data. We present an accurate probabilistic noise model for RGB-D cameras and most of our work builds on it.



# TABLE OF CONTENTS

|                                                                                 | Page |
|---------------------------------------------------------------------------------|------|
| Chapter 1: Introduction . . . . .                                               | 1    |
| 1.1 Thesis Statement . . . . .                                                  | 4    |
| 1.2 Specifics of Our Segmentation Problems . . . . .                            | 4    |
| 1.3 New Functionality for Mobile Robots . . . . .                               | 5    |
| 1.4 Motion Cues for Segmentation . . . . .                                      | 7    |
| 1.5 Thesis Overview . . . . .                                                   | 8    |
| Chapter 2: Background . . . . .                                                 | 10   |
| 2.1 RGB-D Sensing . . . . .                                                     | 10   |
| 2.2 3-D Scene Reconstruction with RGB-D Cameras . . . . .                       | 13   |
| Chapter 3: RGB-D Sensor Noise Modeling . . . . .                                | 21   |
| 3.1 An RGB-D Sensor Noise Model . . . . .                                       | 22   |
| 3.2 Memory-Saving Approximation . . . . .                                       | 28   |
| 3.3 Estimating Per-Pixel Depth Uncertainty . . . . .                            | 28   |
| 3.4 Modeling Observation Uncertainty . . . . .                                  | 30   |
| 3.5 On Adapting the Measurement Model to Time-of-Flight Depth Cameras . . . . . | 32   |
| Chapter 4: Object Segmentation in Static Scenes . . . . .                       | 35   |
| 4.1 Introduction . . . . .                                                      | 35   |
| 4.2 Related Work . . . . .                                                      | 36   |
| 4.3 Object Segmentation with Change Detection . . . . .                         | 38   |
| 4.4 Using Information from Many Scenes . . . . .                                | 53   |
| 4.5 Object Segmentation from Motion with Dense Feature Matching . . . . .       | 56   |
| Chapter 5: Object Segmentation in Dynamic Scenes . . . . .                      | 69   |
| 5.1 Introduction . . . . .                                                      | 69   |
| 5.2 Related Work . . . . .                                                      | 69   |
| 5.3 RGB-D Flow: Dense 3-D Motion Estimation Using Color and Depth . . . . .     | 70   |
| 5.4 Segmentation from Human Demonstration . . . . .                             | 87   |

|             |                                                 |     |
|-------------|-------------------------------------------------|-----|
| 5.5         | Summary                                         | 94  |
| Chapter 6:  | Online Object Segmentation and Modeling         | 95  |
| 6.1         | Introduction                                    | 96  |
| 6.2         | Related Work                                    | 97  |
| 6.3         | Aligning Frames to a Map                        | 99  |
| 6.4         | Online Change Detection                         | 105 |
| 6.5         | Online Mapping and Segmentation                 | 105 |
| 6.6         | Other Sources of Segmentation Information       | 117 |
| 6.7         | Efficiency                                      | 117 |
| 6.8         | Summary and Future Work                         | 117 |
| Chapter 7:  | Active and Interactive Object Segmentation      | 119 |
| 7.1         | Introduction                                    | 119 |
| 7.2         | Active Segmentation: Causing Camera Motion      | 120 |
| 7.3         | Interactive Segmentation: Causing Object Motion | 146 |
| 7.4         | Summary                                         | 152 |
| Chapter 8:  | Object Discovery                                | 153 |
| 8.1         | Introduction                                    | 153 |
| 8.2         | Related Work                                    | 154 |
| 8.3         | Multi-Scene Analysis                            | 156 |
| 8.4         | Evaluation                                      | 162 |
| 8.5         | Summary and Future Work                         | 169 |
| Chapter 9:  | Summary and Future Work                         | 170 |
| 9.1         | Motivation                                      | 170 |
| 9.2         | Contributions                                   | 170 |
| 9.3         | Future Directions                               | 172 |
|             | Bibliography                                    | 176 |
| Appendix A: | Pseudocode for RGB-D Sensor Measurement Models  | 192 |
| A.1         | Accurate Model                                  | 192 |
| A.2         | Approximate Model                               | 195 |

## ACKNOWLEDGMENTS

Thank you foremost to my advisor, Dieter Fox, who has put a lot of effort into my professional development for an extended period of time. Thank you to my other advisors in grad school, Xiaofeng Ren and Steve Seitz, each of whom has also worked with me beyond being on my committee and helped me in substantial ways. Thank you to the many professional mentors I've had during grad school, whom I refrain from naming in the interest of privacy. Thanks go also to those who have read my paper and thesis drafts at various points, who have improved my writing greatly. Again no names here.





## Chapter 1

### INTRODUCTION

Service robotics is a new and rapidly expanding field. The term means robots that can operate in environments populated mostly by humans and that can help those humans. This entails their being able to perform various tasks, the most studied of which are probably map building, localization, navigation, and identification and manipulation of movable and otherwise changeable aspects of the environment. The most common such aspects found in robotics research to date are movable *objects*, connected regions of material, the identification of which is the focus of this thesis.

Suppose that when robotic butlers finally become commercially available, you order one, have it shipped to your apartment in an urban high-rise, take it out of its shipping crate and turn it on. Let us consider what tasks it should be capable of performing right away. For one thing, it ought to be able to dust all your furniture. This task requires knowing the geometric boundaries of objects, but not objects' identities, in order to dust hidden surfaces and to avoid breaking small objects while moving them or moving near them. For another, it ought to be able to find objects that are unique in geometry and/or color even if it doesn't know their names. If you only own one yellow object with green markings, any reasonably intelligent robot should be able to carry out an instruction like "please bring me the yellow box with green text" (as a proxy for something like "please bring me the toaster oven box"). Another class of tasks involves memorizing shape and color attributes of objects. If a robot is given, say, a pile of silverware and a picture of a table setting on a white background, it ought to be able to reproduce several table settings on your dining table; that instruction might be worded something like "please copy this picture six times at these six locations", with each location indicated by a simple gesture such as two fingers placed on the table. None of these tasks involves knowing the names of objects (in the object search example, the human uses object names "box" and "text", but the robot doesn't need them); each involves only being able to represent objects separately from their surroundings. Computer vision experts call this capability *object segmentation*, and it is the goal of this thesis. Autonomous

object segmentation would provide a basic level of competence for autonomous robots in human-centric environments, without which service robots would be more a danger than a help.

Other tasks robot butlers have to do, such as retrieval of named objects or object search by functionality, require information about the identities of objects present in the scene, or identities linked to location information of objects. The goal of *object recognition*, a well-studied subfield of computer vision, is to determine the identity (in the form of an *object class*, such as “coffee mug” or “car”, or an instance identifier, such as “Evan’s water bottle”) of the object represented by a given pre-segmented piece of visual data. Given views of an object correctly segmented from raw visual data (images or 3-D reconstructions), object recognition can identify the object; this requires at least approximate segmentation. A related subfield, *object detection*, locates all objects of types it knows about in given visual data and identifies the objects’ instances or classes. Usually this involves a computationally expensive search over image regions to find those that most resemble pre-built models of known object instances or types; such a system is called a “sliding-window detector”. Object recognition and detection provide reasonably accurate and possibly efficient [113] recognition of a large number of known object or object class models. Typically, object recognition and detection systems are based on training a visual classifier on a set of labeled training examples [126, 156, 114]. There are two major reasons for which automatic segmentation is useful in providing these training examples.

Firstly, recognition and detection performance improve as training set size increases, and acquiring labeled training examples from humans is expensive in time and money. Any automation of the creation of these models is helpful. Either we can manage to create labeled examples using some sort of geometric reasoning or we can at least create models and ask humans to help with the labeling. This is still better than requiring that humans make the models from scratch. Secondly, in an uncontrolled environment there will continually be objects that aren’t already known by the recognition system; an autonomous robot will need to acquire training examples during operation. Segmentation is a preferable source to recognition for getting these training examples because most object recognition and detection systems don’t model object boundaries as precisely as we do here. Since the introduction of visual point features like SIFT [121], most efficient object recognition systems perform point feature matching and represent an object’s pose in a scene as a set of pixels

near keypoints rather than as an outline (e.g. [65]). Others locate objects as bounding boxes in images by processing rectangular image windows. The PASCAL object detection challenge [53], for example, measures overlap of estimated and true bounding boxes to determine accuracy. Malisiewicz and Efros [126] comment on the gap between the ability of detection systems to identify what object classes are present and their inability to identify the spatial extent of these classes:

The 2006 PASCAL Visual Object Classes Challenge...consists of two tasks: image classification (does this image contain one or more objects of a given class?) and object detection (find and localize all instances of an object class). Tellingly, the contest results on the classification task were overwhelmingly good, especially using the bag-of-words techniques, whereas the performance on the detection task was quite poor. This disparity suggests that the current methods are having a hard time grouping image evidence into coherent objects.

Due partly to this argument, PASCAL now also has a segmentation challenge, in which results must be given as a pixelwise segmentation of an image rather than a bounding box. These segmentations are by object class rather than by instance as in our work, so are not comparable to any of our results. In 2011 the pixel class labeling accuracy reached about 43% overall [54]; we obtain much higher pixel accuracy on a very different problem.

Consider our hypothetical robot butler as a home appliance. Other appliances, such as dishwashers, come with various settings fixed at the factory. Some, like wash time or assumed water temperature, might not be settable outside the factory; others, like a limited selection from wash cycles, are. A robot butler will be orders of magnitude more complex than a dishwasher and will have exponentially more settings that might be changeable by the user. In order for a home user to be able to use the robot at all without spending years teaching it first, some supervised training performed by the manufacturer is reasonable. However, due to the extreme variety of environments domestic robots will face, they will also need some skills usable with no training, for example for segmenting previously unseen objects.

Therefore, the problem we address in this thesis is the segmentation into objects of 3-D scenes. For our purposes *scene* means roughly “the information about empty space and

surfaces in the immediate vicinity of the robot”. We obtain descriptions of scenes from video recordings. Segmentation of a 3-D scene can be defined as determining the geometric boundaries of all movable objects in the scene. This definition holds whether objects are being moved around or not, and applies to all objects that appear in the scene at any time during recording.

### 1.1 Thesis Statement

Our thesis is as follows:

**Recent progress in efficient 3-D scene reconstruction and in sensors that provide depth images make it possible for mobile robots to *autonomously and robustly* segment all movable rigid objects in their environments.**

We will discuss these two enabling technologies for object segmentation in a separate chapter. RGB-D cameras, which combine color cameras with recently developed technology for providing dense depth measurements, have been around for only about five years. They are introduced in sec. 2.1. Recent advances in fast and robust 3-D reconstruction, which allow us to update a map with each new camera frame *before processing the next frame*, are discussed in sec. 2.2. The word “all” here refers to textureless objects: a major focus of our work is to analyze highly textured and untextured objects with the same algorithms.

### 1.2 Specifics of Our Segmentation Problems

Our segmentation techniques are designed for rigid objects only, although we have done work on estimating nonrigid motion. Almost all object segmentation work to date has assumed object rigidity due to the strong spatial regularization that assumption provides. We intend to be able to find all rigid objects in a scene that are large enough for our sensors to distinguish. There is no upper size limit except that imposed by a robot’s strength: if a robot is physically unable to move an object, for most of the robot’s purposes that object is in fact not an object. When objects too large for the robot to move, however, are moved by another entity, we can certainly segment them; see fig. 4.8 for an example.

Importantly, unlike most previous work, we impose no requirement of distinctive visual texture on the objects to be segmented; our methods are tested on, and should be able to segment, such scenes as a white bowl moving across a white table in one direction while a white

box moves in another direction. Also importantly, unlike point-feature-based segmentation methods such as those of Collet et al. [38] or Gordon and Lowe [65], our segmentations are *dense*, providing a full partitioning of the elements into which the scene has been discretized (e.g. pixels or surface mesh triangles). Within the body of work performing dense object segmentation, our methods are better motivated than those of, e.g., Pundlik and Birchfield [142], who perform sparse feature-based segmentation followed by a single-image extension of the segmentation to all pixels. A final point about our segmentation methods is that the information and methods we use theoretically allow our segmentations to be arbitrarily close to the true object boundaries. We don’t see pixel-accurate segmentations in practice very often because they would take a large amount of robot time to verify, but unlike methods based on machine learning or heuristics, our methods are at least intended to be capable of producing correct segmentations of all previously unseen objects in previously unseen environments.

The segmentation problems we solve have the important properties of being *live* and being *embodied*. In most cases our scenario is that there is a robot present at the scene. This gives us the opportunity to perform *active vision* by deciding where to move our camera to get a more informative view than the current one or to measure variables that we haven’t yet, as well as the opportunity for *interactive vision*, in which the robot touches the environment to, for example, move an object to see a new part of it. Such problems are significantly different from those tackled by previous work on object segmentation of street scenes as viewed by a car (e.g. [114]), other outside urban scenes (e.g. [6]), and non-close-range indoor scenes (e.g. [163]). The segmentations in these methods aren’t really intended for object instance segmentation and manipulation, but more for class identification. The outdoor scenes mentioned in these papers also include some regions of stuff (as opposed to things [58]). Segmentation is reasonably well defined for stuff, but it isn’t very similar to object segmentation: essentially, stuff can’t be picked up and moved around in one piece.

### 1.3 New Functionality for Mobile Robots

In addition to improving trained vision systems such as object recognizers and detectors, having working object segmentation allows us to add various completely new functionality to our robot systems. For example, we can represent a 3-D map with its static and dynamic

parts. Geometric mapping of static scenes is a well-explored problem (see, e.g., [172, 5, 167]), and one of our goals is to build on static mapping by representing a 3-D map with separate static background and dynamic (movable) objects. This has been done many times before (e.g. [18, 71, 155]) but we do it in a more general way, with more representation detail and more efficiently than previous work. Furthermore, in ch. 6 we build these maps online, while the camera is still present in the scene, which has not been done previously.

We make 3-D models of all the objects we segment. As mentioned before, accurate object models improve object recognition, but they also help in the related task of *object discovery*, the clustering of partial views of objects seen by different sensors and/or at different times to decide which represent the same physical object. For example, Kang et al. [95] use an iterative alternation between segmentation and object discovery to improve both. Object models are also useful for robot manipulation planning, most of which assumes complete geometric models of all objects involved [62, 158, 50]. A very important application of object models that doesn't involve autonomous systems at all is human visualization, for purposes such as teleoperated human-robot cooperation [67], in which communication bandwidth is limited, and virtual home remodeling [35], in which the main purpose of the system is aesthetic. Humans rely in large part on visual information, and unclear representations of remote or virtual environments reduce our ability to use this information effectively. This has been known to cause accidents during teleoperation at such locales as the Fukushima Dai'ichi nuclear plant in Japan [22]. The accident described therein was due to the unsatisfactory visual representation available to the operator.

The term *semantic mapping* means associating any non-geometric information with geometric locations, or eschewing geometric accuracy in favor of topological maps (e.g. [170]). One of the most common types of information associated with geometric regions is object class and identity labels. These can be associated with rooms [102] or with more localized areas [141, 107]. Object segmentation can be used to inform semantic mapping of what regions of the scene are objects and should have object labels. Both Pronobis et al. [141] and Koppula and Saxena [108], for example, use objects as some of the nodes in a graph over semantic labels in the scene, and so need to learn about the set of objects.

## 1.4 Motion Cues for Segmentation

For very simple purposes like pick-up-and-place tasks, rigid objects can be considered to be defined by independent motion: separate objects are those that don't always move in the same way at the same time. This is a simplistic view of objects, but in these early stages of object-based mobile robotics it's an appropriate definition of objects, and it's the definition we use.

It's reasonable for the notion of an object to depend on the capabilities of the robot, so that a robot consisting of a camera and a manipulator arm, for example, is only interested in objects it can both see and manipulate. The robot we use most of the time has a hand on the end of an arm, so in this work we usually consider objects light enough to be moved by the hand. We also *de facto* ignore transparent objects since our sensor is unable to see them reliably.

Once we are using movability as a measure of objectness, it is natural to use motion as a cue to find objects. Objects can move over short or long periods of time; they can move while the robot is present or while the robot is absent, or a mixture of the two; they can move at the instigation of the robot or at the instigation of another agent (probably a human). We intend to use all of these types of motion cue to help segment objects.

We always rely on motion information as much as possible and avoid making strong assumptions about object appearance or using complicated appearance priors. We do always need non-motion cues eventually due to sensor noise and the difficulty of spatially regularizing information across untextured surface regions. However, we only add this information to our algorithms after finding out how far motion alone can take us.

Over the last few years robotic vision systems have gotten much better at detailed and very fast 3-D reconstruction of static scenes using volumetric approaches running on the GPU [134, 73, 188]. Our work can be considered as extending these systems in multiple ways to deal with motion of rigid objects. The goal of this thesis, then, is a life-long [105] mapping system that understands and models rigid objects.

One benefit of using motion information is that our segmentations are not intended to be approximate, and when a segmentation is known to be incorrect, we have some quantification of its uncertainty. This benefit is not achievable with methods not intended to give accurate segmentations, such as bounding-box methods commonly used in 2-D computer vision, or

with methods that use entirely machine learning or heuristics for segmentation, such as [129, 7].

### 1.5 Thesis Overview

RGB-D cameras, which provide high-resolution images in both color and distance to a surface, have recently become available at consumer prices. The color and depth channels have different noise characteristics. In ch. 2.1 we discuss these sensors. Ch. 2.2 shows how we use RGB-D cameras to build geometric reconstructions of scenes from video, a capability that the rest of the thesis uses and builds on. In ch. 3 we present a sensor noise model for RGB-D cameras that takes advantage of their multimodality. Much of the rest of the thesis builds on this model as well.

Our work focuses on methods for using change over time (over multiple visits to a scene or during one visit) to autonomously segment human-scale rigid objects from each other and the background. We take advantage of recently introduced sensing technology and of having a robot that can affect the configuration of objects near it. We have proposed three algorithms for using different temporal cues to segment objects. In ch. 4 we show two methods for segmenting objects from long-term unobserved motion. First we use our sensor noise model to do change detection between two RGB-D scenes, and use the results to perform object segmentation. This method is unable to find correspondences between objects in different scenes, so we also demonstrate a segmentation method using visual descriptor matching, which is able to perform object correspondence.

One limitation of these segmentation methods is that they require that objects be immobile during video recording. Ch. 5 presents two segmentation algorithms that work on video in which objects move. First we discuss a method for estimating dense nonrigid 3-D motion (scene flow) from two views of a scene containing moving objects, and we use this motion estimation to segment the objects. Then, in order to provide a segmentation tool with better accuracy than any of the automated methods we have shown so far, we introduce a method to make use of a (restricted) human interaction with an object that is intended to demonstrate the object to the system.

Segmentation algorithms that use data containing moving objects can be used in an online system, in which segmentation information is updated periodically during the robot’s



presence in a scene. Other work needed to create such a system includes making mapping, segmentation and object modeling efficient enough for online use as well as updating object models with information derived from segmentation. In ch. 6 we demonstrate an online (although not real-time) system that performs reconstruction, object segmentation and modeling of any number of rigid objects.

Assuming the environment is also populated by humans, some objects will move often regardless of what the robot does. However, there will always be some objects that humans don't move very often or that rarely appear without surrounding clutter. These scenarios motivate *active* and *interactive* object segmentation, in which the robot literally takes matters into its own hands and causes the camera or objects to move. Ch. 7 presents our experiments with active and interactive segmentation techniques. We have integrated active mapping into our online object segmentation system to allow the robot to control what data it receives.

The segmentation techniques mentioned so far create object models but can't decide which models are of the same objects (except in limited cases: sometimes we do perform tracking during a scene). An *object discovery* technique we present in ch. 4 tells us which views represent the same physical object. This allows us to track objects across long periods of time and would allow us to collect information like the distribution of objects in space (an object-independent version of which is used in [9] to create a probability map over the static environment of where objects tend to be located) and patterns in which objects are in the same place at the same time (which, along with information about which objects are touched by a person when, can be used for activity recognition [146]).

## Chapter 2

### BACKGROUND

#### 2.1 *RGB-D Sensing*

The term *RGB-D* has recently been coined for sensors that provide both color and range (red/green/blue/depth) information. In the last two decades the most common techniques for acquiring RGB-D data have been (1) the combination of color cameras with laser range finders (such as those made by SICK [2], Hokuyo [122] and Velodyne [117]) and (2) passive stereopsis, in which a sparse set of approximate depth measurements are acquired from two or more images taken simultaneously from calibrated color cameras by using pattern matching of small color image windows. In the last few years there have been major advances in active stereoptic sensors, which project light in known patterns and sense it to determine distance, and in shrinking time-of-flight sensors, which compute depth from the time a light pulse takes to leave and return to the sensor. Currently multiple companies (e.g. [45, 189, 42, 41]) market centimeter- or millimeter- accurate high-resolution depth cameras, at least for indoor use. An *RGB-D camera* combines a color and a depth camera and outputs registered images from both at each frame. This is significantly more depth information than was available just a few years ago from small-form-factor depth sensors such as [44, 43]. The RGB-D cameras used in this research are the PrimeSensor (the PrimeSense research prototype of what would become the Kinect), the Microsoft Kinect and the Asus Xtion Pro Live, all of which use more or less exactly the same PrimeSense technology. The active stereo system of the PrimeSense cameras uses infrared (IR) light so as to avoid interfering with the color camera. The sun tends to overpower the low-power projectors in these devices, so the depth cameras are mainly useful indoors.

The basic principle used by active stereo systems is difference in *parallax*, the difference in position of scene elements in the images taken from two slightly different camera locations. The parallax of an object is a function of its distance from the camera: the farther an object is from the camera, the less the object's position in the image changes as the camera physically moves. Active stereo systems use a projector and a camera between which there

is a known, fixed physical offset, rather than a single camera undergoing known motion. Texture patches with known location in the projected image are identified in the camera image, and the differences in the two locations of each patch are used to estimate its depth. Additionally, while PrimeSense has not released technical information about the PrimeSensor’s depth measurement system, it is likely [125] that the depth camera uses an astigmatic lens, which has different focal lengths in the x and y directions. A phenomenon commonly used for depth estimation in computer vision is that blurriness increases with depth from the camera. The astigmatism in the PrimeSensor’s lens causes the amount of blurring of the known projected pattern to additionally differ in the x and y directions, making blur an even more useful cue for depth. This cue is combined with parallax and with spatial smoothing to produce final depth estimates. All this processing is performed on the device.

The most common alternative to active stereo in depth cameras is time-of-flight (TOF) technology, which uses any of several kinds of information about the return of an emitted light pulse to estimate distance from the sensor to the returning surface, which is assumed to be located in the direction of travel of the beam. Most surfaces reflect at least some of the emitted light back to a sensor located close to the light source. The most common depth sensor for vision research before the advent of the Kinect was the SwissRanger [43], which uses the phase shift of a frequency-modulated signal to determine depth uniquely within a range of around 10 meters. The recently released Microsoft Kinect 2 probably uses a similar method, as it is likely based on TOF cameras made by Canesta [189], which was purchased by Microsoft in 2010. By contrast, laser range finders, which tend to have the largest maximum depth ranges of all depth measurement devices used in robotics, use a laser pulse for each measurement. Such a system uses either a time-coded aperture to capture the part of the beam that returns within a given interval or some sort of temporal integration to determine when light starts and stops returning.

While RGB-D sensing always provides strictly more information than color-only cameras, RGB-D data are especially useful for segmentation for three reasons. The depth difference across some object edges is very large, and this is a much stronger indicator of object edges than any color-only cue. Secondly, areas of little color difference aren’t very strong cues for object boundaries; for example, white dishware piled inside a white cabinet will show very little evidence for boundaries in a color-only image but will contain many edges with large depth discontinuities. Thirdly, 3-D shape data are qualitatively different from 2-D

color data for the purpose of 3-D scene reconstruction, as they can be treated as noisy measurements of exactly the variables that are estimated in geometric reconstruction.

Balanced against these advantages are some drawbacks of depth camera measurements. Depth cameras are much noisier than color cameras, essentially for two reasons. First is that many pixels don't have depth readings. The projector and sensor in an active stereo system have to be physically offset, of course, so they don't see all the same surfaces; as a result many pixels in the depth sensor's view are shadowed from the texture projected by the system. Also various physical phenomena of surfaces such as transparency and specularity prevent the projected texture from being reflected toward the camera. The second reason is that today's depth cameras are much lower-resolution than color cameras. For example, the depth camera in the Kinect appears (in unpublished investigations conducted at Intel Research Seattle) to have approximately an eighth the resolution that the depth images output by the Kinect have. As a result, to approximate depth at each color-image pixel, the device performs a large amount of smoothing. (See [106] for more details.) This results in visually pleasing, approximately correct, but still noticeably noisy depth measurements. Since the noise in depth measurements comes from different sources than those affecting color (e.g. lighting, moving shutters), noise modeling for the two is very different. We focus on modeling depth-camera noise.

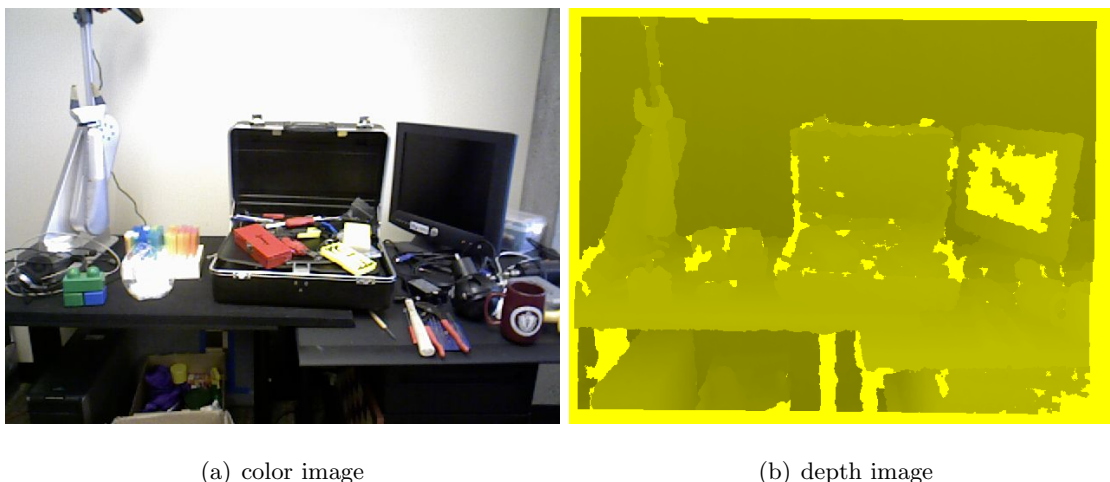


Figure 2.1: a sample color and depth frame from a Kinect. The scene is a toolbox on a desk.

Fig. 2.1 shows an RGB-D frame from a Kinect. The scene is an open toolbox on a desk

next to a robot arm. The depth map is color coded with a linear map from yellow at  $z = 0$  to black at  $z = z_{max} = 10$ . We use  $z = 0$  for pixels without valid depth measurements. The thick yellow band at the right edge of the depth map is pixels that are visible from the IR camera but not from the IR projector. (Again, [106] provides a good explanation.) The yellow areas at the top, bottom and left edges of the depth map are an artifact of Kinect processing; the PrimeSensor and Xtion devices provide readings in these regions. The rest of the invalid depth readings are caused by a combination of nonreflective surfaces (e.g. the monitor screen), excessively specular surfaces, and depth discontinuities (e.g. the left edge of the toolbox), near which stereo sensors all have trouble estimating depth.

A PrimeSensor-based device provides both color and depth images at 640x480 (VGA) resolution at 30 frames per second. Most frames have about 250,000 valid depth values (of a maximum of 307,200). The color and depth image streams are not temporally synchronized by the device, so we use software synchronization methods that essentially throw away frames from one stream that don't have close temporal matches in the other stream. In our experience, almost always frame dropping is caused by software on the machine that receives sensor data, rather than by the RGB-D camera. We usually record RGB-D video, saving a video as a sequence of frames. In all the work in this thesis, both color and depth images are saved with either no compression or lossless compression: we use neither lossy image compression nor lossy video compression on data that will be automatically processed.

## 2.2 3-D Scene Reconstruction with RGB-D Cameras

The data structure on which all the interesting operations in this thesis operate is the *scene*: the geometry of the environment during the shooting of one contiguous segment of video. We assume in most cases that nothing that the camera can see moves at any point while the video is being taken. Each frame consists of approximately 250,000 colored 3-D points, and since temporally contiguous frames see mostly the same surfaces, representing a scene by either raw frames or the concatenation of the point clouds extracted from frames is redundant. Additionally, such representations are unwieldy and noisy. We therefore aggregate sensor measurements of the same scene into 3-D reconstructions. The reconstruction representations we use have the additional advantage of smoothing measurements from multiple frames, in each case within a probabilistic framework.

3-D reconstruction with sensors is a well-studied field. In the computer vision community it is commonly called *structure from motion* or *multiview stereo*, and in the robotics community it's usually called *mapping*. Reconstruction is usually done with either color camera data or distance sensor data (optionally with color data also so that reconstructed surfaces can be colored). Most 3-D reconstruction algorithms assume static scenes: the camera moves in order to view surfaces from multiple angles, but nothing else in the scene moves while the recording is being made. The environment being reconstructed, therefore, does not change over the course of recording, and can be represented in a single data structure. There has also been reconstruction work that makes less restrictive assumptions: that parts of the environment move during recording, for example. Reconstruction of a static environment through which objects are moving, without also modeling the moving objects, is often called *detection and tracking of moving objects* [182]. It is possible to also model the moving objects (as done by, e.g., [71] with distance sensors and [150] with color cameras).

All of our work builds on mapping systems that make use of RGB-D cameras. The work in chapters 4 and 8 was done using RGB-D Mapping [75], a dense mapping system developed for large indoor environments such as buildings. RGB-D Mapping assumes a static scene. At a high level, RGB-D Mapping works with a representation known as *pose graph*: a graph in which each node represents the pose of the camera at the time of a particular video frame. Edges between pose graph nodes specify information about relative poses between different frames; pairs of nodes without edges are those for which we have no direct evidence of relative pose, but are still constrained relative to each other by the rest of the graph, which is a connected graph. In RGB-D Mapping, evidence for guesses at these relative transforms comes from two visual alignment procedures: visual point feature matching (using color data) and point cloud alignment (using color and depth data).

A process known as *visual odometry* provides estimated transformations between pairs of consecutive frames. Visual odometry is performed by a combination of visual point feature matching and dense point cloud registration. We use SIFT [121] to provide image keypoints and local descriptors; we have experimented with more efficient feature detectors such as FAST [149] and descriptors such as Calonder's [29], but we mostly use SIFT because the faster alternatives result in significantly worse reconstructions in our settings. For odometry by feature matching, each visual feature is assigned the depth of the nearest pixel. The 3-D transformation between each pair of time-adjacent frames can be estimated using a

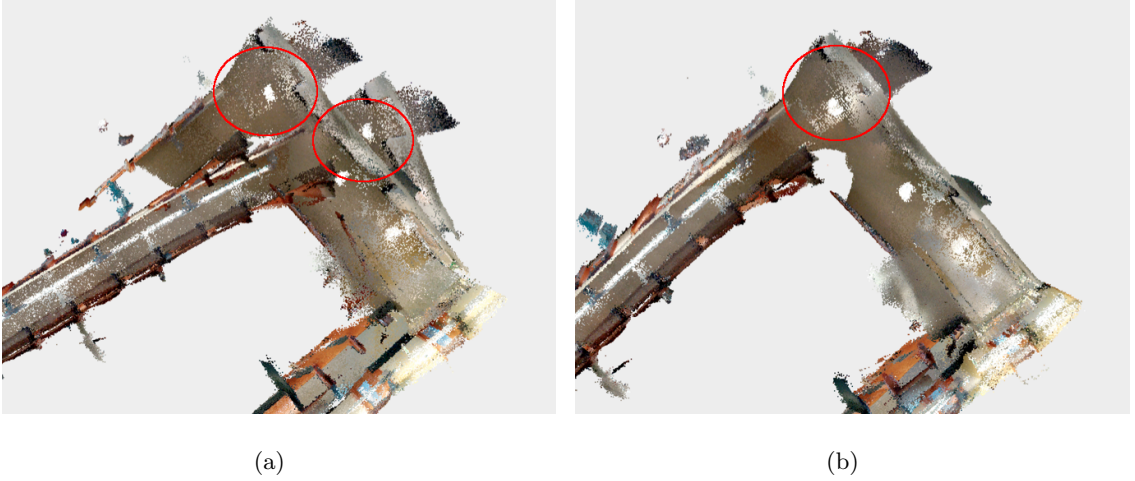


Figure 2.2: views of a 3-D map of an office building reconstructed with RGB-D Mapping (b) with and (a) without loop closure. The circled areas show where the same physical location was reconstructed in multiple places in this supposedly metrically accurate map.

RANSAC [56] algorithm, using Horn’s method [84] to fit models. We optionally follow the feature matching step with the iterated closest points (ICP) algorithm for aligning point clouds [15], which often results in more accurate alignments than those given by feature matching alone but requires an initial estimate (which we get from feature matching). Alternatively we can use a modified version of ICP that optimizes, in addition to the positions of a random subset of all 3-D points in the frame, as standard ICP does, those of matched keypoints. In this algorithm, detailed in [75], the correspondences of most points change at each optimization iteration, as in standard ICP, but those of keypoints produced by the feature detector do not. This stabilizes the optimization and ensures that the estimated transformation does not diverge from that suggested by the feature correspondences alone. In sec. 4.3.2 we demonstrate the improvement the ICP step provides over point feature matching alone.

Visual odometry provides enough edges to connect the pose graph, but due to drift over time and possibly to both algorithmic and sensor bias, the resulting representations are not globally consistent: the same area of the environment seen at different times during recording can be reconstructed multiple times at large intervening distances (see fig. 2.2). A *loop closure detection* module selects pairs of nonconsecutive frames between which edges should



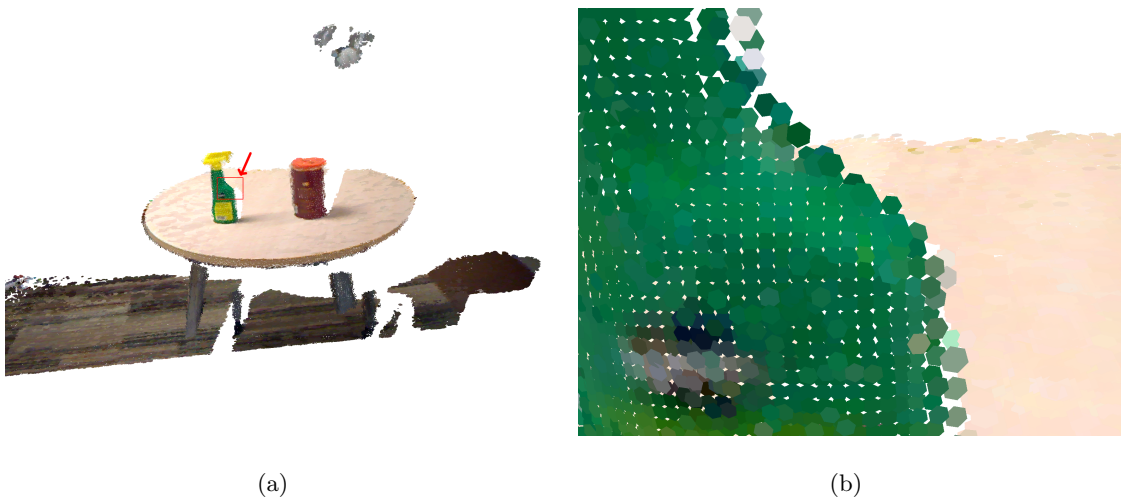


Figure 2.3: a demonstration of surfels. (a) a map, in the form of surfels, made from approximately a hundred RGB-D frames. (b) a zoom of the indicated part of the map, to show how much surfels usually overlap and how smooth surfaces represented as surfels usually are.

be added in the pose graph. For efficiency we select a subset of frames to be *keyframes* (usually either by using a simple heuristic such as selecting every 20th frame or by waiting for enough motion to occur, as estimated through visual odometry, before creating a new keyframe) and only consider adding edges between keyframes. We run visual feature matching on each pair of keyframes. For each pair of keyframes whose feature match count exceeds a constant threshold (whose value is in the low tens), the same RANSAC algorithm as before is used to estimate the 3-D transformation between the frames. Each pose graph edge is described by the transform produced by RANSAC (and ICP when it's used) as well as some uncertainty information about that transform. We then use a pose graph optimizer to globally adjust all camera poses. The constraints used by this optimizer [69] do not encode all the information from feature matching, since the optimizer is only given a single transformation estimate and its covariance, so the optimization may introduce local inconsistencies. Therefore, RGB-D Mapping follows pose graph optimization with a bundle adjustment [104] step, again using 3-D-augmented SIFT descriptors to provide point correspondences. Bundle adjustment, which optimizes both camera poses and the positions of point features that were viewed by two or more cameras, improves local reconstruction consistency without disrupting global consistency.



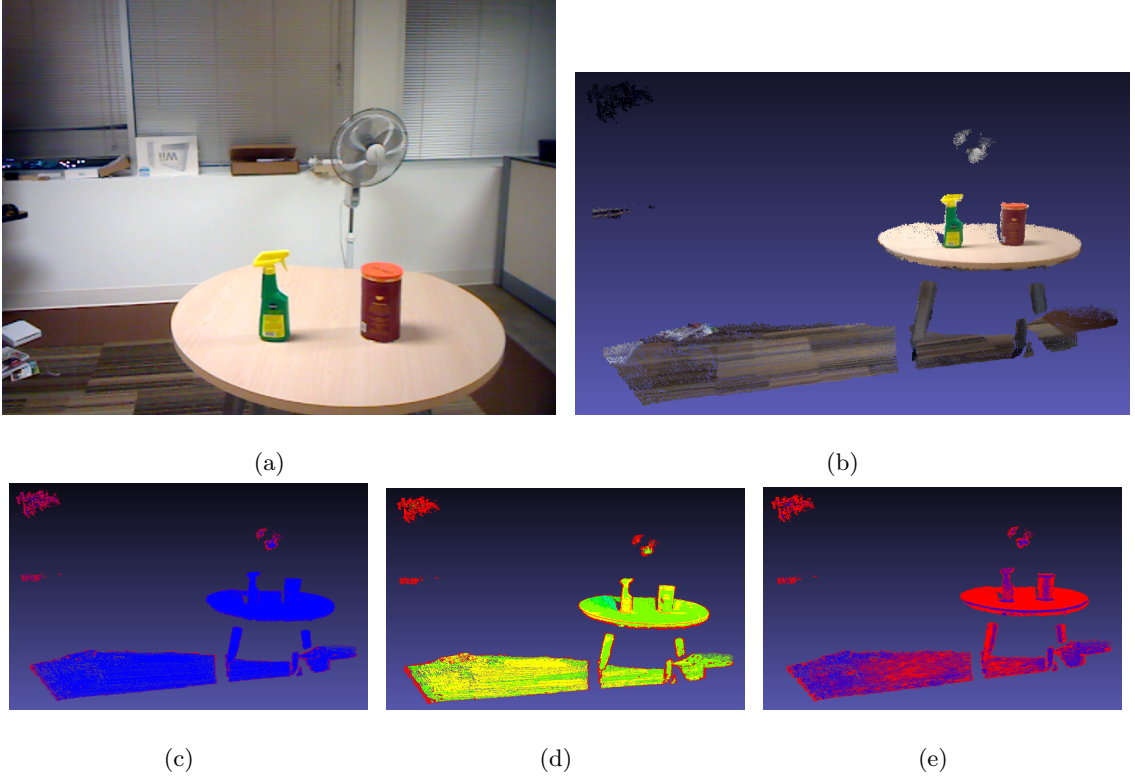


Figure 2.4: a visualization of surfel uncertainty. (a) a video frame. (b) the surfel map of the scene. (c) surfels colored by uncertainty in position: blue (low) through purple to red (high). (d) surfels colored by uncertainty in color (variance of the Gaussian mixture component created with the most data): cyan (low) through green, yellow and orange to red (high). (e) surfels colored by uncertainty in planar orientation: red (low) through purple to blue (high).

Following alignment of frames, RGB-D Mapping aggregates point clouds into *surfels* [139], small circular surface patches that consist of position, orientation (surface normal), radius and color. Each 3-D point from each frame is assigned to a surfel, either an existing one or a newly created one if there is no existing surfel close to the point. Fig. 2.3 shows a surfel map and a close-up to demonstrate how smooth surfaces represented as surfels tend to be and how much surfels tend to overlap. We use the parameterized rules proposed by [139] to add points to surfels in an online fashion. However, under the formulation of surfels that we use, surfel building cannot happen in a truly online fashion: the loop closure step has the ability to change the positions of points, which would normally invalidate

surfel computations in progress. This forces us, when building surfel maps, to run map aggregation after pose graph optimization of the entire video. In RGB-D Mapping we can optionally augment the surfel representation by probabilistically modeling the position, color and orientation of each surfel, keeping uncertainty information as well as the usual MAP values. A Gaussian over position can be updated recursively during reconstruction, and we compute distributions over color (Gaussian mixture) and normal (von Mises) after reconstruction. Fig. 2.4 shows a surfel map and the position, color and normal uncertainty value at each surfel. The map is shown from approximately the viewpoints from which it was captured, so noise tends to increase with distance from the viewer; noise is also high at the edges of surfaces, partly because of imperfect alignment between color and depth pixels and partly due to decreased sample size for averaging at those locations.

One common alternative to patch-based surface models such as surfels is *occupancy grids*, such as are used by [193, 18] in 2-D and [87] in 3-D. Patches have three major advantages over grid representations for dense 3-D reconstruction. One is space efficiency; grids take space exponential in the map dimension (two or three). Surface patches have better spatial resolution than grid models, as surfels can be arbitrarily closely spaced. Patches also provide greater shape detail than grid maps. Surfaces must be algorithmically extracted from grid representations when needed for surface sampling or visualization. The most efficient such algorithm is marching cubes [120], which produces a (continuous but non-smooth) triangulated mesh. More sophisticated algorithms such as Gaussian process implicit surfaces [191] can produce smooth implicit surfaces from grid representations but are not efficient enough for most uses and are not scalable. However, the surfel representation has two major disadvantages. A large number of heuristics are necessary for determining when to add, merge and delete surfels. Additionally, surface-based maps can't include information about *unseen space*, the volume blocked from the camera's view by intervening surfaces. Grid representations are well suited to describe what space is unseen and what space is known to be empty. Because one of the goals of this thesis is to represent as much information as we can discover about surfaces and objects, this last advantage of volumetric mapping techniques is an important one, so the work in chapters 6 and 7 was performed using volumetric maps.

We use the Patch Volumes SLAM system [73] for representing volumetric maps and updating them with RGB-D frames. The Patch Volumes framework represents a scene as one or more truncated-signed-distance-function (TSDF) volumes. Each such *patch volume*

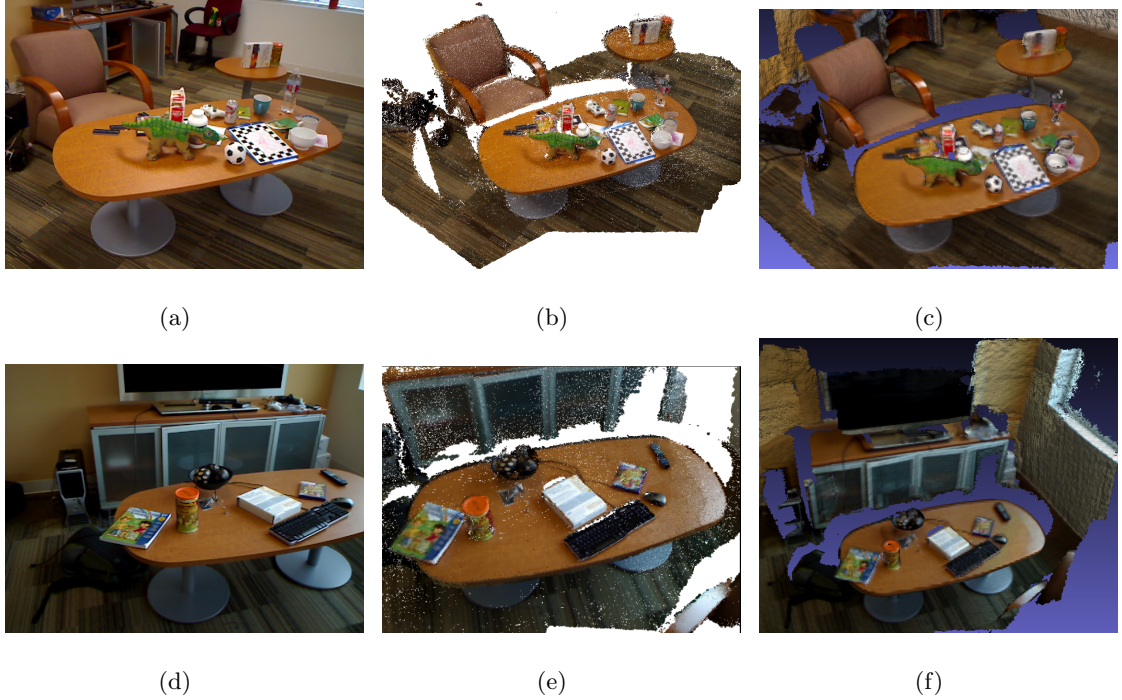


Figure 2.5: a visual comparison of the RGB-D Mapping and Patch Volumes mapping systems. (a) a frame from a video; (b) the map produced for the scene by RGB-D Mapping; (c) the map produced for the scene by Patch Volumes. (d), (e), (f) the same for a different scene. Surfel maps tend to blur details less than volumetric maps if constructed at the same map resolution; notice, e.g., the soccer ball and clipboard in the upper row and the coloring book in the lower row. (Resolution in surfel maps depends on camera distance from viewed surfaces.) Volumetric maps tend to be less noisy than surfel maps at sharp edges; notice, e.g., the chair arms in the upper row and the bottom of the cabinet in the lower row.

consists of four voxel grids that store the signed distance to the nearest surface, a measure of certainty of the TSDF value, the color in each voxel, and a measure of certainty of the color. The certainty weights allow for efficient depth map fusion using the technique of Curless and Levoy ([46]) on the GPU (as introduced by KinectFusion [134]). This depth map fusion obtains smooth, high-quality geometry estimates from multiple noisy depth frames. Color fusion is performed in a similar manner. A new frame is aligned against this surface prediction by iteratively minimizing a dense error function which is the sum over pixels of a depth error and a color error, represented in a sum-of-squares form. This

combined error function provides robust alignment in cases where either depth or color alone provides insufficient constraints on camera motion. The TSDF representation allows for parallel GPU ray casting for surface prediction given a camera pose, and the current scene model can be extracted at any time as a colored mesh by using the marching cubes meshing algorithm [120]. Patch Volumes runs mostly on the GPU and can extend the size of the modeled volume dynamically and automatically as necessary. One improvement of Patch Volumes over RGB-D Mapping specifically is its reduced reliance on point features: it is no longer necessary to have visual texture in many parts of the image in order to align frames.

Fig. 2.5 compares the outputs of RGB-D Mapping and Patch Volumes on two videos used in sec. 4.3. Neither representation dominates the other. Further, the detail of a volumetric map depends heavily on its resolution. We find a 1-cm resolution to produce reasonably nice-looking models for many objects, but as shown in sec. 6.5, decreasing the voxel size can make much nicer models even of objects without very fine detail.

Similarly to our extension to the surfel representation to store uncertainty information about surface samples, it should be possible to extract probabilistic information about the depth and color at a given pixel of a camera viewing a Patch Volumes map, although this has not been implemented to date. This should be achievable on the GPU, like most Patch Volumes operations. One way in which this could be done is a rendering operation, in which voxels are processed independently, that accumulates counts and values at pixels rather than creating fragments with depth at each pixel as is done in standard rendering. An alternative is the slightly more approximate approach of Krainin et al. [109], who make use of the particular formula used to calculate voxel weights in the volumetric reconstruction of [46]. This weight formulation is also used by Patch Volumes, so the second option may be the most convenient for us. In this way we should be able to preserve our ability to obtain probabilistic models of surface properties in a map while obtaining the benefits of GPU parallelization of mapping.

Both RGB-D Mapping and Patch Volumes are state-of-the-art and *usable* 3-D reconstruction systems. Mapping technology is maturing, making it possible for us to use it as a building block in more complex systems. In particular, in chapters 4 and 8 we run mapping as the first step of a segmentation pipeline.

## Chapter 3

### RGB-D SENSOR NOISE MODELING

Most vision algorithms that use sensor readings can benefit from the use of uncertainty information about those measurements. For example, patch-based map aggregation methods can use depth uncertainty information from individual readings to update depth uncertainty information per surfel; volumetric map aggregation methods can use it to decide what regions of space a surface might exist in; and geometric comparison methods can use it to decide how highly to weight the matching error of a particular surface patch. The work in this thesis uses sensor measurement uncertainty information not only in order to represent uncertainty in maps but for change detection, alignment of 3-D shapes, and outlier rejection within various procedures.

We present here a detailed and accurate probabilistic measurement model (a.k.a. “noise model”) for RGB-D sensors using active stereo. Probabilistic measurement models are common in robot localization [59] and mapping [171]. This is natural: generative models are appealing for work with sensors because often we can accurately describe the process of measurement generation, and Bayesian models appeal when we need to combine many sensor readings. Hähnel et al. add dynamic object estimation to SLAM and use expectation-maximization to label each measurement static or dynamic [71]. Kim et al. probabilistically fuse information from multiple depth sensors and cameras for dense 3-D reconstruction [101]. We use a generative sensor measurement model with similar sensor modalities to those of [101], but discard some of their independence assumptions. Our model is based on the *beam-based* model of Fox et al. [59], who model sonar and laser scanner readings of a space expected to be mostly empty but populated by a small number of people. Like them, we discretize space, although our bins have size on the order of a millimeter rather than centimeters. We expect many spatial bins to be empty in many circumstances, but the model parameters we choose differ slightly from those of Fox et al. due to our having greater expectation of the space in the vicinity of the camera being free (in addition to much more significant differences due to differing sensor characteristics).

First we present the full probabilistic measurement model for RGB-D sensors, which is based on that of [59] but incorporates color and surface orientation measurements. We also present an approximate model that requires no storage of precomputed values, which saves multiple gigabytes of memory in our case. We discuss modeling measurement uncertainty at the cost of increased runtime, as well as the potential for a speedup from parallelization. We mention the relationship between our work and a data-driven depth sensor noise model published since our work was done.

### 3.1 An RGB-D Sensor Noise Model

The color and depth measurements from RGB-D sensors don't have the same level of reliability. Apart from the various reasons discussed in sec. 2.1 for missing depth measurements, the uncertainty in depth measurements from any stereo system increases approximately quadratically with depth (consider eqn. 11.1 of [169] evaluated at disparities  $d$  and  $d + 1$ ). Finally, Kinect depth measurements at nearby pixels are highly correlated not only due to the spatial extent of an object usually being much greater than that of a single pixel, but because the Kinect in particular greatly smooths its depth measurements (see sec. 2.1). Because of all these factors, it's more important when working with depth measurements than with color to have a model of sensor uncertainty. We have introduced one [76] that we believe to be the most accurate available for RGB-D cameras, although for computational efficiency it treats the measurements at nearby pixels as independent.

A sensor noise model gives a distribution over sensor readings given true values of the measured quantities. We make use of dense depth information to locally estimate surface normals and define a virtual measurement  $z = \langle z_d, z_c, z_\alpha \rangle$ , where  $z_d$  is the depth value of the pixel,  $z_c$  the color, and  $z_\alpha$  the orientation of the detected surface. Our sensor noise model for RGB-D cameras gives distributions over measured depth, color, and surface orientation given the true, or "expected", depth, color and surface orientation of the surface seen at a pixel, which we model as another virtual measurement  $z^* = \langle z_d^*, z_c^*, z_\alpha^* \rangle$ :

$$p(z \mid z^*) = p(z_d, z_c, z_\alpha \mid z^*). \quad (3.1)$$

As will become clear in sec. 4.3.3, we also want to consider the case in which we do not have an expected value of the measurement. That is, we need a model of all sensor measurements,

aggregating over all possible values of expected surface attributes:

$$p(z) = p(z_d, z_c, z_\alpha). \quad (3.2)$$

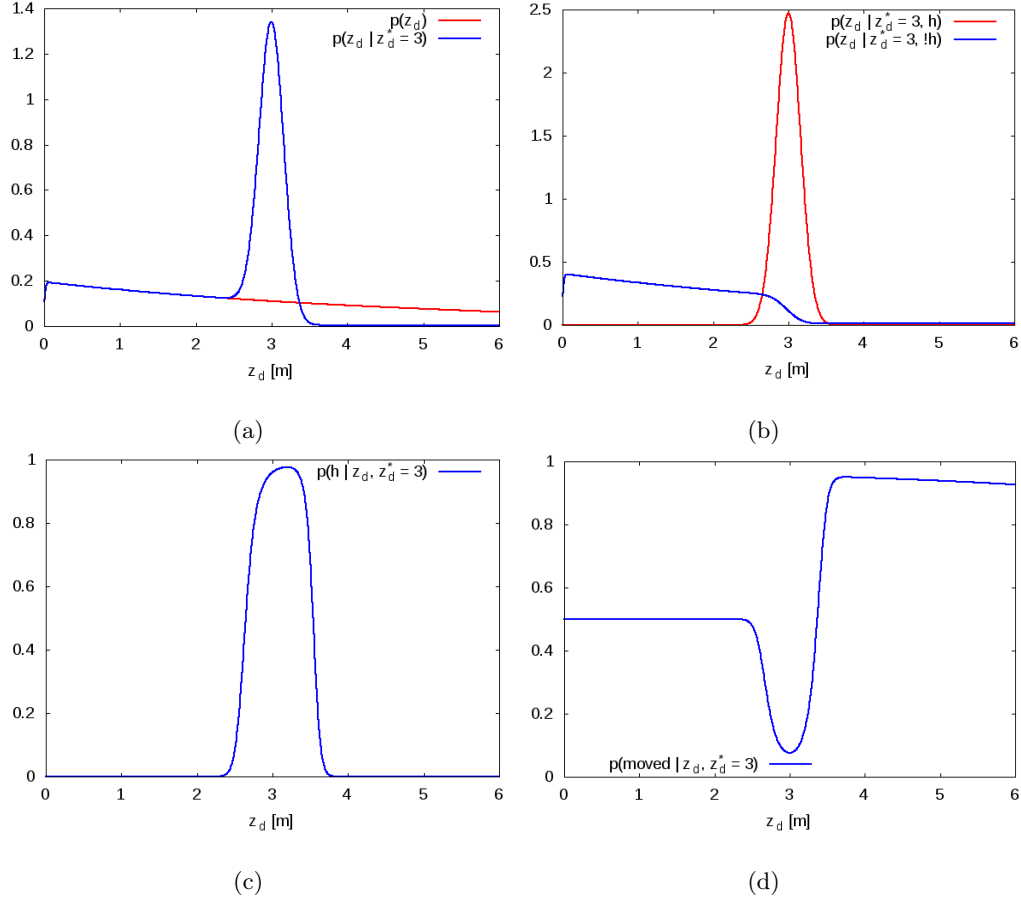


Figure 3.1: visualizations of our sensor noise model for qualitative examination. (a) probability of a depth measurement  $z_d$  given no knowledge about the environment (red) and given a model in which the closest surface patch along the pixel beam is at 3 m (blue). (b) probability of measuring distance  $z_d$  given that the known surface caused (red) or did not cause (blue) the measurement. (c) probability that the surface at  $z_d^* = 3$  caused the measurement  $z_d$ . This probability serves as a weighting function for the mixture components of the color and orientation models. (d) probability that the expected surface is missing given a depth measurement  $z_d$ . The model parameters used to make these plots are different from those used in our experiments.



### 3.1.1 No Scene / Surface Information

We now model an RGB-D pixel measurement  $z$  if there is no prior information about surfaces in the scene. We assume that in the absence of any surface information, the depth, color, and orientation measurement components are independent:

$$p(z) = p(z_d, z_c, z_\alpha) \approx p(z_d)p(z_c)p(z_\alpha) \quad (3.3)$$

Sensors that can be modeled as shooting beams in various directions, such as laser scanners and sonar systems, are referred to as *beam-based*. Following models developed for beam-based range sensors [171, 60], we model the distribution over depth measurements given no information about surfaces as a mixture between an exponential and a uniform distribution:

$$\begin{aligned} p(z_d) &= \begin{pmatrix} w_{\text{short}} \\ w_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{short}}(z_d) \\ p_{\text{rand}}(z_d) \end{pmatrix} \\ p_{\text{short}}(z_d) &= \begin{cases} \eta \lambda_{\text{short}} e^{-\lambda_{\text{short}} z_d} & \text{if } z_d \leq z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \\ p_{\text{rand}}(z_d) &= \begin{cases} \frac{1}{z_{\text{max}}} & \text{if } z_d < z_{\text{max}} \\ 0 & \text{otherwise} \end{cases}, \end{aligned} \quad (3.4)$$

where  $p_{\text{short}}$  is the probability that the beam detects a surface before reaching maximum range, and  $p_{\text{rand}}$  is a uniform distribution over the measurement range.  $w_{\text{short}}$  and  $w_{\text{rand}}$  are mixture weights and  $\lambda_{\text{short}}$  and  $\eta$  are the scale parameter of the exponential distribution and its normalizer, respectively. An example for this mixture distribution is represented by the red line in fig. 3.1(a).

Given no map of the environment, we model the color and orientation distributions  $p(z_c)$  and  $p(z_\alpha)$  as uniform distributions over the color and orientation spaces.

### 3.1.2 Known Scene / Surface

We now assume that a map of expected surfaces is available. Given the pose of the depth camera, we can use ray tracing to determine the distance, color, and local orientation of the closest surface along the pixel direction. This information can be encoded in an *expected* measurement  $z^* = \langle z_d^*, z_c^*, z_\alpha^* \rangle$ . Conditioning on the expected measurement and separating



the individual components of the pixel measurement, we apply the chain rule:

$$\begin{aligned}
p(z \mid z^*) &= p(z_d, z_c, z_\alpha \mid z^*) \\
&= p(z_d \mid z^*)p(z_c \mid z_d, z^*)p(z_\alpha \mid z_d, z_c, z^*) \\
&\approx p(z_d \mid z_d^*)p(z_c \mid z_d, z^*)p(z_\alpha \mid z_d, z^*) .
\end{aligned} \tag{3.5}$$

In the last step we assume that the orientation  $z_\alpha$  is independent of the color  $z_c$ . This model is similar to that of [101], which combines three cues—depth, color, and image gradients—for 3-D reconstruction. However, while they assume independence of the components, we explicitly model the dependences between color, orientation, and depth. As we will show below, this results in more accurate models since, given a surface patch, the distance provides additional information about the expected orientation and color.

We now derive models for the three components. The model for the depth value  $p(z_d \mid z_d^*)$  is almost identical to that of existing beam-based models. Here, we stay close to the one given by Thrun and colleagues [171], who model  $p(z_d \mid z_d^*)$  as a mixture of four distributions. (We can also use the more correct but less intuitive model of [59]; see sec. 3.2.) We use three of their components. We do not explicitly model maximum-range measurements, since these correspond to uninformative noise in the cameras we use and can be ignored without loss of accuracy. Incorporating maximum range if necessary would be straightforward.

$$p(z_d \mid z_d^*) = \begin{pmatrix} w_{\text{hit}} \\ w_{\text{short}} \\ w_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_d \mid z_d^*) \\ p_{\text{short}}(z_d \mid z_d^*) \\ p_{\text{rand}}(z_d \mid z_d^*) \end{pmatrix} \tag{3.6}$$

$p_{\text{hit}}$  represents the case in which the measurement hits the expected surface, and the other two components are just like those modeling unknown surfaces, with the exponential distribution cut off at the expected distance. The distribution for the first case is a Gaussian centered at the expected distance:

$$p_{\text{hit}}(z_d \mid z_d^*, z_\alpha^*) = \begin{cases} \eta \mathcal{N}(z_d; z_d^*, \sigma_{\text{hit}}^2(z^*)^2) & \text{if } z_d \leq z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Here  $z_{\text{max}}$  is the maximum measurement range and  $\eta$  is a normalizer.  $\sigma_{\text{hit}}^2$  is the distance measurement noise, which is a function of  $z_d^*$ , of the obliqueness  $z_\alpha^*$  of the viewing angle to

the expected surface, and of the error in stereo depth estimation as a function of depth (we use a stereo noise model since our depth camera technology is based on stereo). Intuitively, the more obliquely the camera views a surface, the less accurate the measurement of that surface will be; we model this by

$$\sigma_{\text{hit}} = \frac{\sigma_0(z_d^*)}{\sin(\pi/2 - \theta)},$$

where  $\theta$  is the angle between the viewing direction and the expected surface's normal and  $\sigma_0$  is a value calculated using expected depth only. (This depth-uncertainty model is intended for use with single RGB-D frames; later in this thesis we will discuss depth uncertainty modeling for aggregated map data.) An example for the mixture distribution is given by the blue line in fig. 3.1(a). Fig. 3.1(d) visualizes the posterior  $p(m \mid z_d, z_d^*)$  resulting from ignoring color and orientation.

We now turn to the models for the color and orientation components of a measurement, respectively  $p(z_c \mid z_d, z^*)$  and  $p(z_\alpha \mid z_d, z^*)$ . The color model obviously should depend on  $z_c^*$ , the color of the expected surface. However, the additional conditioning on  $z_d$  allows us to use the depth value to determine whether the measurement was actually caused by the expected surface or by some other surface (due to moving occluders, for example). To do so, we introduce a binary random variable  $h$  whose value is whether the expected surface is the surface that caused the measurement.

$$\begin{aligned} p(z_c \mid z_d, z^*) &= p(z_c \mid z_d, z^*, h) p(h \mid z_d, z^*) \\ &\quad + p(z_c \mid z_d, z^*, \neg h) p(\neg h \mid z_d, z^*) \end{aligned} \tag{3.8}$$

follows directly by conditioning on  $h$ . We can model the first component of this mixture by a Gaussian with mean at the expected color:

$$p(z_c \mid z_d, z^*, h) \sim \mathcal{N}(z_c^*, \sigma_{\text{col}}^2) \tag{3.9}$$

We model the other mixture component in (3.8),  $p(z_c \mid z_d, z^*, \neg h)$ , by a uniform distribution over the color space.

To determine the weights of these two mixture components, we apply Bayes' rule to

$p(h \mid z_d, z^*)$  and  $p(-h \mid z_d, z^*)$ , yielding

$$p(h \mid z_d, z^*) \propto p(z_d \mid h, z^*)p(h \mid z^*) \quad (3.10)$$

$$p(-h \mid z_d, z^*) \propto p(z_d \mid -h, z^*)p(-h \mid z^*) \quad (3.11)$$

with the same constant of proportionality in both. Here,  $p(z_d \mid h, z^*)$  and  $p(z_d \mid -h, z^*)$  are special cases of the distance model given in (3.6):  $p(z_d \mid h, z^*)$  is the Gaussian component  $p_{\text{hit}}$ , and  $p(z_d \mid -h, z^*)$  is a mixture of the other two components  $p_{\text{short}}$  and  $p_{\text{rand}}$ .

$p(h \mid z^*)$  in (3.10) can be computed as  $\int_0^{z_{\text{max}}} \frac{p_{\text{hit}}(z_d \mid z^*)}{p(z_d \mid z^*)} dz_d$ . Example distributions for  $p(z_d \mid h, z^*)$  and  $p(z_d \mid -h, z^*)$  are given in fig. 3.1(b). Plugging these distributions into the weighting function (3.10) results in the plot shown in fig. 3.1(c). Again, this distribution serves as a weighting function for the mixture components in the color model (3.8). As a result, the measured color  $z_c$  should be similar to the expected color  $z_c^*$  if the depth measurement  $z_d$  is close to the surface distance  $z_d^*$ , and uniformly distributed otherwise.

The derivation of the orientation model  $p(z_\alpha \mid z_d, z^*)$  is analogous and omitted for brevity. We model orientation by a von Mises-Fisher distribution, a common choice of distribution over the sphere because it has a closed-form pdf [17]. We use parameters  $\mu = z_\alpha^*$  and  $\kappa = 60$ .

One way to balance evidence from these three components is to change the effective number of observations [171]. We use the reweighting

$$p(z \mid z^*) = p(z_d \mid z_d^*)^\alpha p(z_c \mid z_d, z^*)^\beta p(z_\alpha \mid z_d, z^*)^\gamma. \quad (3.12)$$

The values of  $\alpha, \beta$  and  $\gamma$  depend on the relative accuracy of the components in the data being compared. For example, surfel maps produced by RGB-D Mapping tend to have relatively high position uncertainty, medium color uncertainty and relatively low orientation uncertainty. By contrast, the surface maps produced by meshing volumetric representations such as those from Patch Volumes tend to have relatively low position uncertainty and relatively high color and orientation noise. For experiments with surfel maps we usually use  $\alpha = 1, \beta = 25, \gamma = 5$  to allow the color and normal components to override the depth component when the observed and expected depths are similar. These numbers are an educated guess, and results are robust to a wide range of values. For experiments with volumetric maps we usually use  $\alpha = 1$  and  $\beta$  and  $\gamma$  values between 0 and 0.5.

### 3.2 Memory-Saving Approximation

The model of [59], which properly takes distance effects into account by processing each depth bin before the next, requires a preprocessing step to build up tables to be referenced at runtime. This step takes a relatively small amount of time (1 to 10 minutes) when run in parallel, but the resulting table requires (for Kinect-like sensors and a depth bin width of 5mm) around two gigabytes of memory. Pseudocode for the preprocessing and runtime computation needed by this sensor model is given in alg. A.1. If we instead use the approximate model based on [171] that is discussed above, runtime is not appreciably faster but very little memory is needed. Pseudocode for this approach is given in alg. A.2.

Our algorithm for the full model precomputes three tables: one giving the probability of each possible observed depth given that the previously seen surface has moved, one giving the probability of each possible observation given that the previous surface has not moved, and one giving the probability that the expected surface is the surface that caused the measurement, given that the expected surface has not moved. Most of the computation at runtime consists of reading these tables.

The computation for the color and surface normal components of the model is the same in the two models, since the approximation is entirely in the more complicated depth component. In the approximate model, we sum values of the three distributions  $p_{\text{hit}}$ ,  $p_{\text{short}}$ ,  $p_{\text{rand}}$  mentioned above, approximating the integral of each distribution over the width of a depth quantization bin by a single sample in the middle of the bin. This approximation can cause problems for extremely peaked distributions; in particular it is advisable to put a lower bound on the variance of the Gaussian distributions  $p_{\text{hit}}$  used with this model.

Fig. 3.2 compares this approximate model to the full model on a pair of scenes. The figure shows the result of change detection using each model (see sec. 4.3). The two results are very similar, as desired.

### 3.3 Estimating Per-Pixel Depth Uncertainty

The variance of the Gaussian  $p(z_d \mid h, z^*) = \mathcal{N}(z_d; z^*, \sigma_d)$  is not specified by the model presented above; it must be provided by the user. Initially we used a per-pixel model for this depth uncertainty, with a parametric form fit to some sample data we gathered from

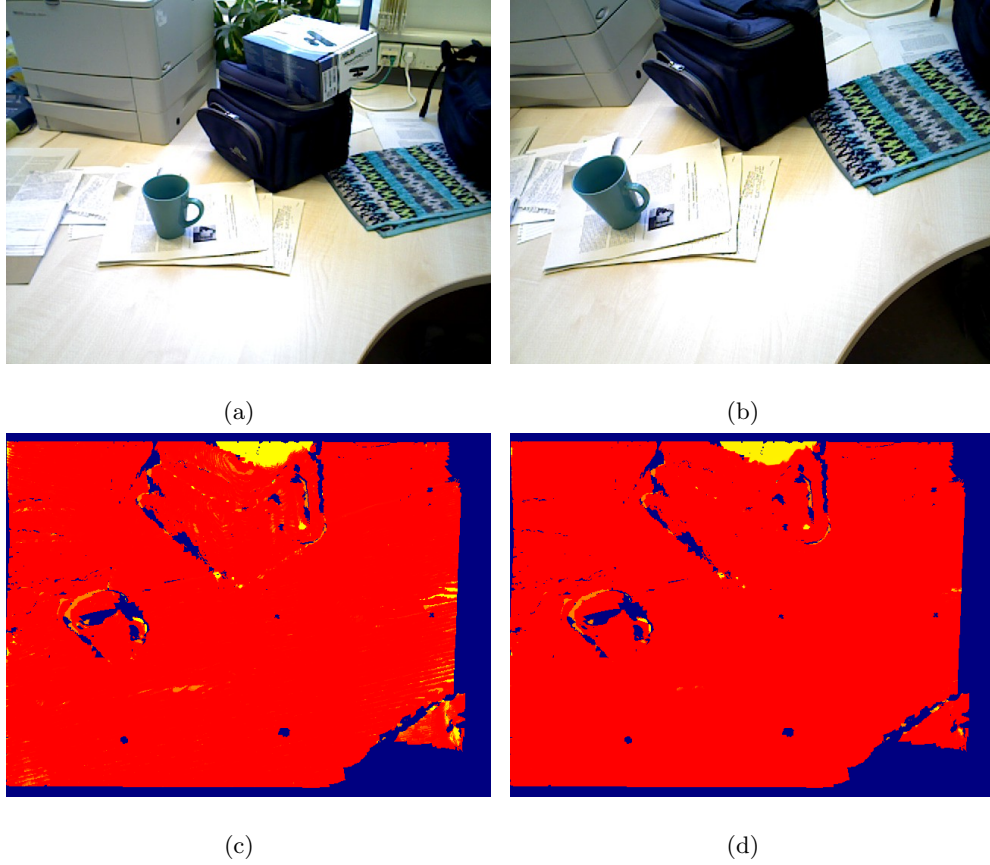


Figure 3.2: a comparison of measurement model variations. (a) a frame from the “expected” scene; (b) the frame used as the “observation”. The following images show the difference in the results of the measurement models with and without expected surface. Images are red where the observed measurement is much more likely under the model with expected surface, yellow where it is much more likely under the model without expected surface, and shades of orange in between. Dark blue pixels are those at which either the expected or observed depth reading is invalid. (c) using the full model of alg. A.1; (d) using our memory-saving approximate model. The two results are almost identical.

several RGB-D frames:

$$\sigma_d(z) = \frac{z^2}{\frac{1}{.00285} + z}. \quad (3.13)$$

Recently, Izadi et al. have performed a more systematic fitting of such a per-pixel model, using images of planar surfaces seen perpendicularly at various depths [135]. Whereas our

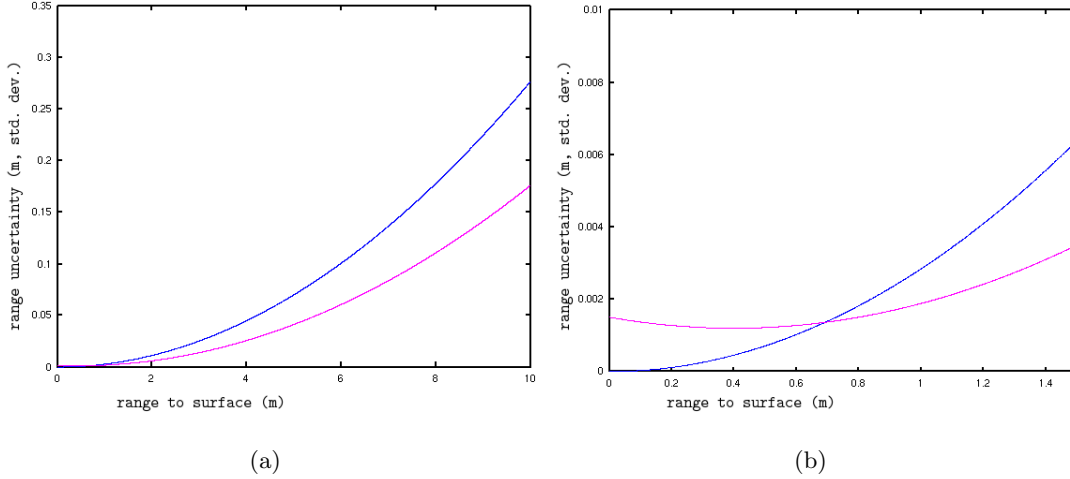


Figure 3.3: our (blue) and Izadi et al’s (magenta) [135] depth uncertainty models as a function of the depth at a single pixel. Their model was developed on a Kinect for Windows and ours on a Primesensor; the hardware may be slightly different. Neither sensor actually returns valid values under about .3 meters, so neither model’s values there matter unless it’s used with depth data from a source other than a Kinect, such as a previously built map rendered from an arbitrary view. The second view (b) zooms in on the small- $z$  regime where a crossover happens.

depth uncertainty formula is intended to be valid for depths .3 to 10 meters (the full range of the Kinect hardware), Izadi et al.’s model is intended for depths in .4 to 3 meters, the official range of the Kinect in near mode. The formula used by Izadi et al. is

$$\sigma_d(z) = .0012 + .0019(z - .4)^2. \quad (3.14)$$

Both these models break down for surfaces seen highly obliquely from the camera. We set an upper limit for surface obliqueness of  $70^\circ$ , although this is merely an educated guess; Izadi et al. use  $60^\circ$  as the maximum following a systematic experiment. Fig. 3.3 plots both formulae for depth uncertainty as a function of single-pixel depth. The two formulae produce similar values, providing large-scale validation for our formula.

### 3.4 Modeling Observation Uncertainty

These depth uncertainties are hard to estimate and have a noticeable impact on model output. There are two relatively simple ways to reduce the errors caused by bad depth

uncertainty estimates. One is to slightly increase our depth uncertainty estimates for all pixels. Another is to include uncertainty in observed depth in the model. Including the observation uncertainty reduces the model’s certainty (i.e. moves the output probability toward .5) in cases where the observed depth value is not very certain. This is useful both for points with large observed depth and for points near depth boundaries, which can be misaligned in the image plane. Uncertainty in expected depth is already included via  $p(z_d \mid h, z^*)$ , but so far we have treated observed depth values as having very little uncertainty. Adding observed-depth uncertainty to the table-based model of alg. A.1 would require a separate table for each quantized value of depth uncertainty, so we instead add this feature to the approximate model of alg. A.2.

We replace  $p(z_d \mid z_d^*)$  in eqns. 3.5, 3.10 and 3.11 with a marginalization over the observed depth  $z_d$ : instead of  $p(z_d \mid z_d^*)$  we use  $p(z_d \mid z_d^*, \sigma_d, \sigma_d^*)$ , which is defined as

$$\begin{aligned}
 p(z_d \mid z_d^*, \sigma_d, \sigma_d^*) &= \int_{z_d^*} \int_{z_d} p(z_d \mid z_d^*, \sigma_d^*) dz_d dz_d^* \\
 &= \int_{z_d^*} f_{exp}(z_d^*) \left[ \int_{z_d} f_{obs}(z_d) p_{bin}(z_d, z_d^*) dz_d \right] dz_d^* \\
 &\approx \int_{z_d^*} f_{exp}(z_d^*) [B f_{obs}(z_d^*)] dz_d^*.
 \end{aligned} \tag{3.15}$$

We have made the dependence on uncertainty information explicit:  $\sigma_d$  and  $\sigma_d^*$  are the standard deviations of  $z_d$  and  $z_d^*$  respectively, and  $p(z_d \mid z_d^*, \sigma_d^*)$  here was previously referred to (in sec. 3.1) as  $p(z_d \mid z_d^*)$ .  $p_{bin}(z_d, z_d^*)$  is an indicator function for whether the two depth values are quantized to the same value by the sensor: for bin size  $B = 1$  mm (a camera-specific value),  $p_{bin}(z, z') = \begin{cases} 1, & |z - z'| < \frac{B}{2} \\ 0, & \text{else} \end{cases}$ .  $f_{obs}$  quantifies observation noise: we use it to average over values that might have been observed. Similarly,  $f_{exp}$  is used to average over possible values of the expected depth. Both are modeled as Gaussians: for pixel  $p$ ,  $f_{obs}(z) = \mathcal{N}(z \mid z_d(p), \sigma_d(p))$  and  $f_{exp}(z) = \mathcal{N}(z \mid z_d^*(p), \sigma_d^*(p))$ .

We have found that taking nonlocal depth information into account by increasing estimated uncertainty values at pixels near large depth discontinuities and pixels with invalid depth readings improves the model. We increase uncertainties at pixels whose distance to a depth discontinuity or invalid reading is small. We represent invalid depth readings with an almost arbitrarily selected mean and a very large variance. We therefore compute  $\sigma_d(p)$  using alg. 3.1 for generating depth uncertainty values for RGB-D frames, which replaces

the less sophisticated model of eq. 3.7.  $\sigma_d^*(p)$  is in general computed from aggregated map information using different algorithms.

---

**Algorithm 3.1** computation of frame depth uncertainties using nonlocal information.  $\sigma_{max}$  is an uncertainty value assigned to invalid pixels; we use 1m.

---

**for each** pixel  $p$  **do**

$\sigma_d(p) \leftarrow StereoNoiseAtDepth(z(p))$

$I \leftarrow$  the set of pixels with invalid depth values or large depth discontinuities

Compute the Euclidean distance transform  $D(p)$  of pixels to  $I$

**for each** pixel  $p$  such that  $D(p) < 3$  **do**

$\alpha \leftarrow \frac{D(p)}{3}$

$\sigma_d(p) \leftarrow e^{\alpha \log(\sigma_d(p)) + (1-\alpha) \log(\sigma_{max})}$

---

The changes to alg. A.2 needed to accommodate this uncertainty modeling are minimal. There must be an additional parameter *obsZSigma* to function COMPUTE, and the uses of  $p(z_d | z_d^*)$  on lines 11 and 12 must be replaced with  $p(z_d | z_d^*, \sigma_d, \sigma_d^*)$ .

Fig. 3.4 gives an example comparison of the approximate sensor model with and without observed-depth uncertainty modeling. In this single-frame example the changes are small but noticeable; over the course of a long video, they can make a large difference.

### 3.5 On Adapting the Measurement Model to Time-of-Flight Depth Cameras

Active stereo is not the only popular technology used in depth cameras. There are also multiple companies selling depth cameras using various time-of-flight (TOF) technologies. The new version of the Kinect uses TOF technology, although since Microsoft has recently acquired multiple makers of TOF cameras, it is not clear what TOF method exactly is used in the Kinect 2 [190]. Regardless of what physical phenomenon is made use of, our RGB-D sensor model is easily adapted to such cameras, as the resulting readings look similar for all types of TOF cameras. The most important change that would need to be made is for the user to specify a depth uncertainty that is constant or periodic with a small amplitude, rather than increasing quadratically, in depth value. The other changes I suggest are due to the interpixel correlation being much lower in TOF cameras than in the Kinect and probably other active stereo systems. Because of this and limits of TOF technology, the depth values are noisier in TOF systems, so uncertainties should be higher than for active



stereo and it is more important to spatially regularize the results of any kind of processing using outputs of a sensor model than it is when using active stereo. See sec. [4.3](#) for an example of such regularization on top of Kinect data.

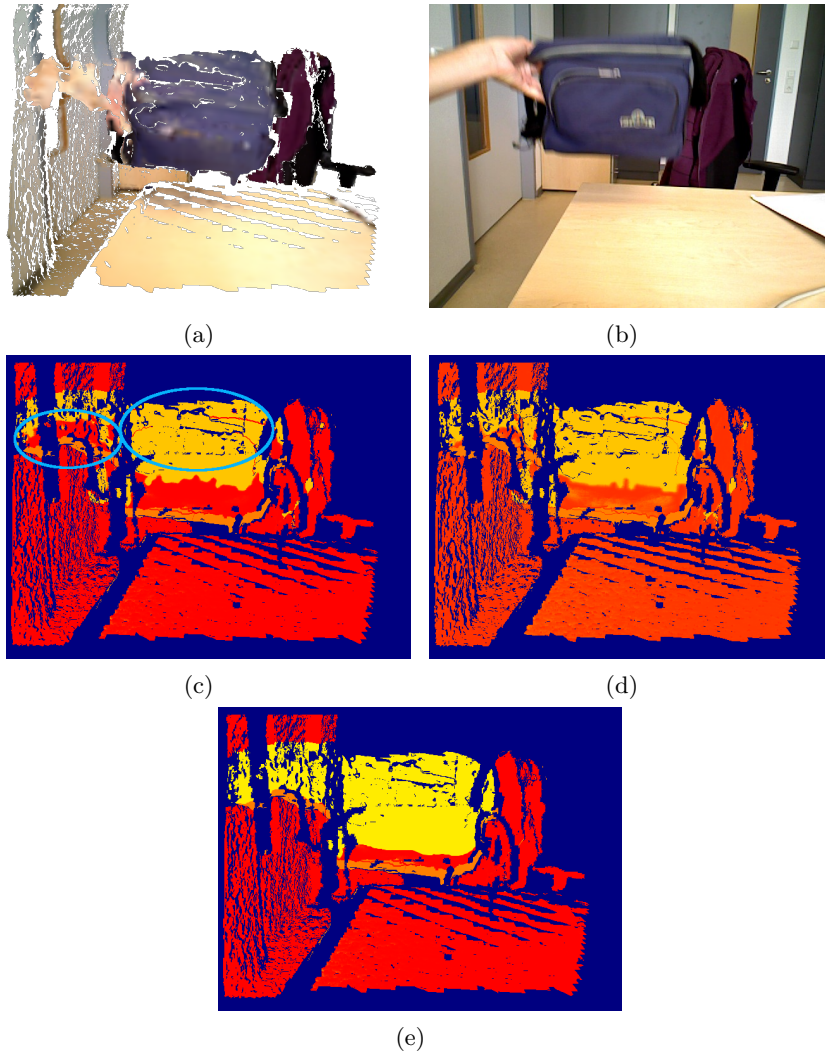


Figure 3.4: comparing our approximate sensor noise model (d) with and (c) without observed-depth uncertainty modeling. As in fig. 3.2, we show the result of a comparison of the models that do and do not assume an expected surface. (This is change detection, which will be discussed in sec. 4.3.) The two scenes being compared (which are, in this case, a map created from previous frames of the current video and the current video frame, respectively) are shown in (a) and (b). The lunchbox is being moved. A manually edited approximate ground truth is given in (e). It isn't pixel-accurate; what's important is that it gives the approximate outline of the regions that should show up as changed.

(d) improves on (c) both in the increase in the yellow area on the hand and in the decreased thickness of the red lines on the lunchbox. Both areas are indicated in (c).

## Chapter 4

## OBJECT SEGMENTATION IN STATIC SCENES

**4.1 Introduction**

Having described our sensors and dealt with modeling their measurements, we turn to object segmentation. As mentioned before, for our purposes object segmentation is the problem of determining the boundaries of rigid objects in a 3-D scene. In addition to providing object models for all kinds of visualizations, object segmentation is very useful in attacking various other situated vision problems, such as object discovery [77] and object manipulation [96].

We can classify segmentation techniques in the fields of robotics and vision as *passive*, *active* or *interactive* according to the kind of actions that are available to the segmentation system. Passive techniques can only use the data that are given to them: the motion of the camera and the objects to be segmented is determined elsewhere. Active techniques can move the camera but cannot manipulate objects. For example, a camera mounted on a pan-tilt unit with electronic control could make use of active vision techniques. Finally, interactive vision techniques assume the existence of an agent that can manipulate the scene on demand, such as a human user or a robot with a kinematic arm. This thesis discusses techniques of all these kinds, but in this chapter we focus on passive techniques.

Segmentation can be *static*, involving only information from a single scene or measurement, or it can use information from sensor readings at multiple times. The most important cues we get from having sensor readings at different times involve motion. Both camera and object motion have been used to provide segmentation cues. It is impossible to get perfect segmentation without motion cues: no combination of static cues can always find object boundaries. Essentially this is because objectness is defined by properties of movement: an object consists of surfaces that always move “together” in some mathematically definable way. Situated vision (having a robot that’s able to manipulate the scene) is also necessary to provide reliably non-heuristic segmentation: even machine learning (given passive vision data) can’t generalize to all segmentation problems, because objects that are separate in some scenes can be glued together in others but look exactly the same. Further, in the case

of scenes crowded by objects close together, most boundaries won’t be visible (although there has been work on estimating them by using learned knowledge about common object sizes and shapes [89]), so camera motion or object motion will be necessary if making full object models is a goal.

## 4.2 *Related Work*

Segmentation is usually done statically, partly for simplicity and partly because static segmentation works well in some cases. It has to be done so when the data are arbitrary sets of photos from different people at different times, but it’s also commonly done statically when run on robots that have physical access to their environments.

The gPb system [7], one of the most popular black-box segmentation systems available today, finds likely boundaries by spatially regularizing over local information consisting of a learned combination of features engineered for the boundary finding problem, the most useful of which are half-disk features, which compare the values of some single-point feature when aggregated over the two sides of a disk centered at the point in question. This approach has recently been extended to learned rather than engineered features [145]. Another approach is segmentation by fixation, intended to model the human visual system. These techniques find points likely to be inside objects, then look outward in the image to find likely boundaries. Mishra and Aloimonos [129] use MAP inference in a Markov random field (MRF) over points in polar space to perform this boundary finding. Carreira and Sminchisescu [30] avoid the seed-point-finding step of this approach by generating a large number of seeds and spatially regularizing similarly for each set of seeds. The main human interaction needed is the step of choosing one of the many results generated this way as the final segmentation. One method that takes advantage of RGB-D data is that of Collet et al. [37], who generate many segments using color data only, use depth data to cull most of them, then rank the remainder according to “objectness” criteria defined over whole segments rather than small local regions.

Because scene segmentation without motion is difficult, most work has made restrictive assumptions. Biswas et al. estimate object shape models and object-cluster memberships for pre-segmented object instances in 2-D occupancy grid maps [18]. They assume the number of objects is known and that each connected component of occupied cells is one object. Wolf

and Sukhatme perform SLAM for long periods of time by maintaining two occupancy grids, one for static and one for movable objects, to identify cells occupied by movable objects at any given time [193]. Connected components is also the standard approach to segmenting objects in 3-D; for example, [153] assumes all objects are on a table, fits a plane to the table and takes each large remaining connected component of 3-D points to be an object. Several authors have segmented objects by fitting geometric solids to point clouds using the method of [157]. This method was developed to approximate a very dense cloud with a small number of models; its applicability to object modeling is less obvious, since a shape that can be fit well by a moderate number of geometric solids is generally better explained by a single higher-level model, e.g. a train engine.

Hager and Wegbreit [72] also use 3-D geometric model fitting to segment objects, employing occlusion and support reasoning to track changes in the configuration of movable objects over time. They start with a static map and assume the camera never moves. Because of the lack of multiple hypotheses, their method is susceptible to errors due to partial occlusion and partial object views that aren't clear enough to justify fitting models. This problem is dealt with for a different application by Ruhnke et al., who collect partial 3-D object models, sample range images from each, and match range images using RANSAC over local 2.5-D features to create models of full objects [151]. Here also, each connected component of a point cloud is taken to be one partial object.

Segmentation of object instances, as opposed to classes, is usually done in a batch fashion. Cosegmentation, one of the vision community's terms for instance segmentation in multiple images simultaneously, has been done in supervised [180] and unsupervised [93, 31] fashions. Kang et al. [95] acquire object instances from a large set of images by iteratively clustering segment hypotheses and refining the outlines of these hypotheses, starting with a "soup of segments" [152]. A robot acquiring knowledge while performing its normal duties cannot afford to use off-line methods.

As we have discussed, motion cues as a rule improve segmentation. Sparse feature point matching between two images has been used, followed by dense regularization, to perform dense segmentation [16]; the same technique has been applied to each consecutive pair of video frames [164] with a slightly different regularization step based on an oversegmentation. Given a noisy mapping from scene elements (e.g. pixels in 2-D or surfels [139] in 3-D) to vectors representing motion between two time points, several authors have clustered motion

vectors in various ways to segment one or more objects from a static background. Bhat et al. [16] use point-feature matches to suggest 3-D rigid motions from 2-D images, then cluster pixels using a single-frame segmentation based on graph cuts. Stein et al. [166] use optical flow as an added cue for boundary finding in single-image segmentation of natural images. Ross and Kaelbling [147], given a set of image pairs each containing movement, learn to detect object edges in single images. In ch. 4 we mention some techniques that use *video* of motion for sparse segmentation.

Most motion segmentation methods rely on sparse feature tracking. In our work we specifically try to avoid relying on having very distinctive visual features, in order to be able to work with textureless and largely occluded objects, both of which occur frequently in real scenes. Another reason to work on dense as opposed to sparse segmentation is that while there are multiple good solutions for sparse segmentation, dense segmentation *given* sparse segmentation is not a solved problem, and techniques that don't rely on distinctive sparse features are necessary even for systems that start with sparse segmentation. One previous project that uses dense motion estimation for segmentation is that of Wedel et al. [185], whose output was a labeling of each scene point as moving or static.

We now describe two algorithms for object segmentation using motion cues without requiring human intervention. The first uses change detection, and the second uses local feature matching, to aid segmentation. In ch. 5 we introduce a third method for autonomous segmentation when the scene is not static, using dense motion estimation. These cues are useful in different situations, and none of these segmentation algorithms dominates any other.

### 4.3 Object Segmentation with Change Detection

One motion cue useful in a very general set of cases is change detection, or *scene differencing*. If the time interval between two sensor measurements is large enough, probably some objects will have moved to completely different locations, appeared or disappeared in the meantime. Scene differencing finds these large-scale changes by looking for differences at each scene (scene element, e.g. pixel or surfel).

#### 4.3.1 Introduction

Suppose a mobile robot with an always-on camera makes multiple visits to a location, continuously running SLAM. There is information in the difference between two maps made at different times. We use an RGB-D camera to detect objects by means of their movement in 3-D between two maps of the same location built from videos taken at different times.

At a high level this problem is similar to laser scan registration, for which van de Ven et al. [177] have shown it is possible to jointly infer clustering within scans and association between scans. They point out that the efficiency of inference depends heavily on the number of elements to be clustered and associated. Therefore we need to select a small number of surfaces by combining surface elements and, prior to inference, culling those least likely to be part of movable objects.

Object segmentation in 2-D has been studied by both the computer vision and the robotics communities. As discussed in sec. 4.2, most segmentation work has not used motion (partly because the datasets needed for such work are simpler than ours). In vision, motion cues have been used for object segmentation assuming highly visually textured surfaces. Bhat et al. [16] assume each surface has enough SIFT matches that RANSAC will find a model for it. That isn't the case for us, partly because our objects occupy relatively little of each image; also we wish to handle textureless objects.

Avoiding the problem of segmenting scenes is common due to the difficulty of segmentation without high-level cues [72]. Because of this, most work has made restrictive assumptions. Since we are segmenting unknown objects, we don't have precomputed information such as a database of predefined object models to fit, so we use only low-level cues. In this section we improve on previous segmentation efforts by identifying surface patches likely to have moved between two scenes. We use ray casting with a probabilistic model of sensor measurements that is robust to sensor noise and that enables us to fuse information from multiple sensor modalities.

Probabilistic measurement models are common in robot localization [59] and mapping [171]. This is natural: generative models are appealing for work with sensors because often we can accurately describe the process of measurement generation, and Bayesian models appeal when we need to combine many sensor readings. Hähnel et al. add dynamic object estimation to SLAM and use expectation-maximization to label each measurement static or

dynamic [71]. Kim et al. probabilistically fuse information from multiple depth sensors and cameras for dense 3-D reconstruction [101]. We use a generative sensor measurement model with similar sensor modalities to those of [101], but discard some of their independence assumptions.

Probabilistic sensor models have also been used for change detection. Kaestner et al. [94] use a model of laser measurements to find changes in outdoor scenes. They perform global alignment of laser scans using iterated closest points (ICP) and detect changes using statistical significance testing for each measurement independently.

Our algorithm aligns the frames of each RGB-D video using SLAM, then reconstructs each scene as a dense set of surface elements (Section 4.3.2). Our sensor measurement models (ch. 3) tell us how likely each surface element is to have moved between two scenes. In Section 4.3.3 we employ the model to detect differences between scenes and in Section 4.3.3 we spatially regularize to find large surface regions that may belong to movable objects.

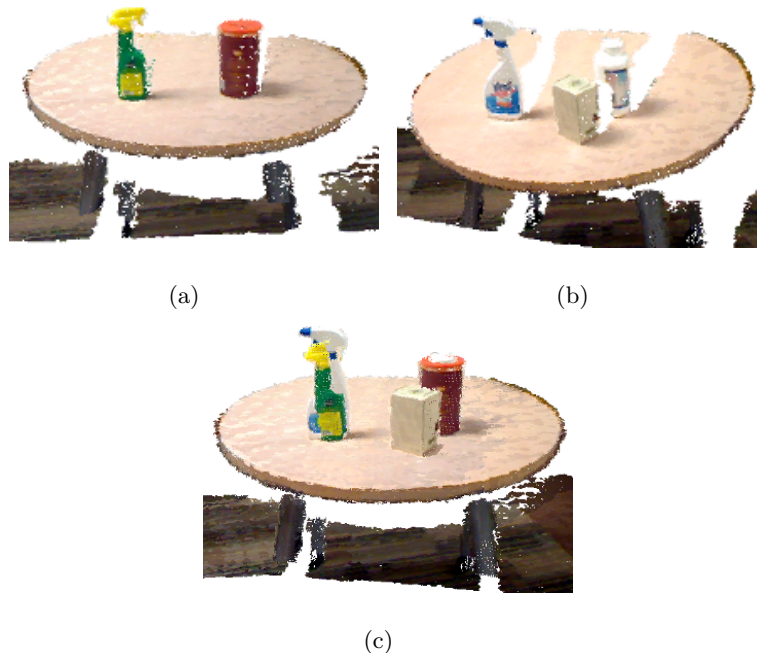


Figure 4.1: interscene alignment. (a), (b) reconstruction of two scenes; (c) their superimposition after alignment.



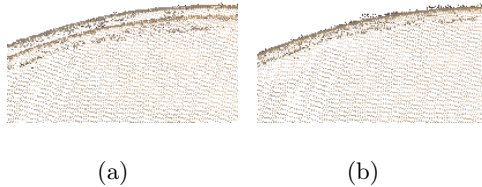


Figure 4.2: SIFT vs. ICP alignment. Zoom into the edge of the table from fig. 4.1: (a) after SIFT RANSAC alignment; (b) after additional ICP alignment.

#### 4.3.2 Scene Reconstruction and Alignment

The inputs to our system are two or more RGB-D videos representing separate visits to the same location. From each sequence we generate a geometrically consistent 3-D reconstruction, or *scene*. Objects can move between visits but are assumed static during each visit.

##### *Scene Reconstruction / SLAM*

We build on RGB-D Mapping [74], a SLAM technique for generating dense 3-D maps. At a high level, RGB-D Mapping performs (1) pairwise frame alignment by visual odometry, (2) loop-closure detection and (3) global path optimization. After all these alignment steps, the mapping system aggregates measurements from different frames into a single surface map represented with surfels. We probabilistically model the position, color and orientation of each surfel, keeping uncertainty information as well as MAP values. We exclude from differencing each surfel that has high uncertainty in any of these attributes.

The rest of our algorithm will require the surface to be representable as a finite set of local samples. We choose to use surfels for this paper, but the algorithm would work equally well with meshes or samples from implicit surfaces.

##### *Scene Alignment*

Assuming moved objects are a small part of each scene, we fit a single rigid transformation to the sets of point features in each pair of scenes using RANSAC. We compute SIFT features, assign each the 3-D location of the nearest depth point in its frame, and transform each image’s features into a scene-global frame. This allows us to compute a transform

between the sets of all features in each scene, which is more robust than the alternative of computing transformations between individual frames of different scenes and picking one to be the global transform. These transforms are inaccurate due to depth noise, so we furthermore run ICP, using the point-to-plane error of [33]. We ignore 30% of the worst-aligned points when calculating error, to account for the movable objects we expect. As an example of our global alignments, Fig. 4.1 shows the superimposition of two aligned scenes. A typical example of the improvement achieved by ICP is shown in Fig. 4.2.

#### 4.3.3 Scene Differencing

Change detection compares a set of samples of one scene’s surfaces at discrete 3-D locations to a set of RGB-D frames. These scene samples can be, for example, 3-D points (3-D locations with associated color and surface orientation), surfels (small planar patches consisting of points plus patch size), voxels, the vertices of any mesh representing surfaces, or samples of an implicit surface function (essentially 3-D points obtained in a particular manner). These representations have different strengths; in particular, points allow the most variation in surface shape over small regions, but voxels are best at representing thin structures.

##### *Pointwise Change Detection*

We perform per-scene-element change detection and smooth the results spatially using a graph over scene elements. The pointwise stage of this process makes use of the two sensor measurement models we developed in ch. 3. The reader will recall models of measurements for the cases when, respectively, the “expected” surface along a pixel ray is known to have moved ( $m = 1$ ) or to have not moved ( $m = 0$ ) between the times of the two measurements:

$$p(z) = p(z_d, z_c, z_\alpha \mid m = 1), \quad (4.1)$$

$$p(z \mid z^*) = p(z_d, z_c, z_\alpha \mid m = 0, z^*). \quad (4.2)$$

We consider the observation  $z$  at a pixel to be independent of observations at all other pixels in the same image as well as of those taken at different times (i.e. from different camera poses during the same recording session). This assumption, similar to that made in naive Bayes classifiers, causes our predictions to in general be too extreme, a tendency that

we combat by inflating the amount of uncertainty in the sensor model parameters beyond that that would accurately model the sensor hardware.

We are now equipped to detect differences between two 3-D maps. Consider a single surface patch in the first scene. Patch  $s = \langle s_x, s_c, s_\alpha \rangle$  is described by its location  $s_x$ , color  $s_c$ , and surface orientation  $s_\alpha$ . Using the alignment technique described in Section 4.3.2, we can determine all camera poses of the second scene relative to  $s$ . Based on these camera poses we identify for each camera frame the pixel that points at that surface patch (if the patch is in the camera's field of view). From the camera pose and the surface descriptor  $s$  we can also compute the expected measurement  $z^*$ . We wish to determine for each  $s$  the probability that it *moved* away from its location in the first scene.

Let  $\mathbf{z}_s$  denote the set of measurements taken in the second scene associated with a specific surface patch  $s$  in the first scene, and let  $\mathbf{z}_s^*$  denote the expected measurements computed from  $\mathbf{z}_s$  and  $s$ . By Bayes' rule,

$$p(\mathbf{m} \mid \mathbf{z}_s, \mathbf{z}_s^*) = \frac{p(\mathbf{m}, \mathbf{z}_s^*)p(\mathbf{z}_s \mid \mathbf{m}, \mathbf{z}_s^*)}{p(\mathbf{z}_s, \mathbf{z}_s^*)} \quad (4.3)$$

$$\approx \frac{p(\mathbf{m})p(\mathbf{z}_s^*)p(\mathbf{z}_s \mid \mathbf{m}, \mathbf{z}_s^*)}{p(\mathbf{z}_s \mid \mathbf{z}_s^*)p(\mathbf{z}_s^*)} \quad (4.4)$$

$$\propto p(\mathbf{m}) \prod_{i=1}^I p(z_i \mid \mathbf{m}, z_i^*), \quad (4.5)$$

where  $I$  is the number of frames in the second scene,  $p(\mathbf{m})$  comes from prior knowledge (we usually use .1 because few of our surfels move) and

$$p(z_i \mid \mathbf{m}, z_i^*) = \begin{cases} p(z_i \mid z_i^*), & \mathbf{m} = 0 \\ p(z_i), & \mathbf{m} = 1 \end{cases}. \quad (4.6)$$

The idea behind (4.6) is that if the surface did not move, the measurement follows the surface-based model (3.5). If the surface patch moved, however, we have no knowledge about what to expect, so we use the prior model (3.3). This prior model encodes knowledge about the overall density of objects expected in scenes.

The pointwise motion probabilities computed for the surface patches of the left scene in Fig. 4.1 are shown in Fig. 4.5(a). The changed objects are detected as high-probability areas, using the combination of depth, color, and orientation cues. Furthermore, the grey areas appropriately represent surface regions that are occluded in the second scene (Fig. 4.1(b)).

However, due to sensor noise and imperfect alignment, the individual motion probabilities are still noisy.

### *Spatial Regularization*

Pointwise change detection gives us the probability of change at each sample point in the scene independently, using only very spatially localized information. We smooth spatially, as well as integrating static segmentation cues like surface curvature, by using these pointwise scores as unary potential values in a Markov random field (MRF) whose nodes represent surface patches. This is how we formalize the notion that if an object moves entirely out of its previous location between two images and moving objects aren’t close together (e.g. if only one object moves at a time), the outline of a changed region is the outline of an object.

Each MRF node has two choices of label: moved and not moved. For efficiency we use only pairwise cliques in the MRF so that we can use graph cuts [23] for inference. The data energy for the MRF is straightforwardly calculated from the output of the pointwise model:

$$E_d(l) = \begin{cases} \log(1 - p(m \mid \mathbf{z}, \mathbf{z}^*)), & l = m \\ \log(p(m \mid \mathbf{z}, \mathbf{z}^*)), & l = \neg m \end{cases} \quad (4.7)$$

For the smoothness energy we use the Potts model, weighted by the curvature of the local region to discourage cuts in low-curvature areas:

$$E_b(s_i, s_j, l_1, l_2) = w_s \begin{cases} \frac{1}{\sqrt{\max(\kappa_i, \kappa_j, \kappa_0)}}, & l_1 \neq l_2 \\ 0, & l_1 = l_2 \end{cases} \quad (4.8)$$

Here  $s_i$  and  $s_j$  are surface patches and  $l_1$  and  $l_2$  their labels.  $\kappa_i$  is the previously computed local curvature at  $s_i$ .  $\kappa_0$  is a constant representing the minimum curvature beyond which all surfaces can be described as “very flat”; we set it to  $1 \text{ m}^{-1}$ . We set the smoothness weight  $w_s$  by optimization over two scene pairs (not the ones visualized in Fig. 4.5 and Fig. 4.6). The result of applying this MRF to the probabilities displayed in Fig. 4.5(a) is shown in Fig. 4.5(b).

### *4.3.4 Experiments*

We evaluate our sensor model and demonstrate the ability to discover and model objects in a tabletop setting. In each experiment, the RGB-D camera was carried by hand to generate partial coverage of the scene.

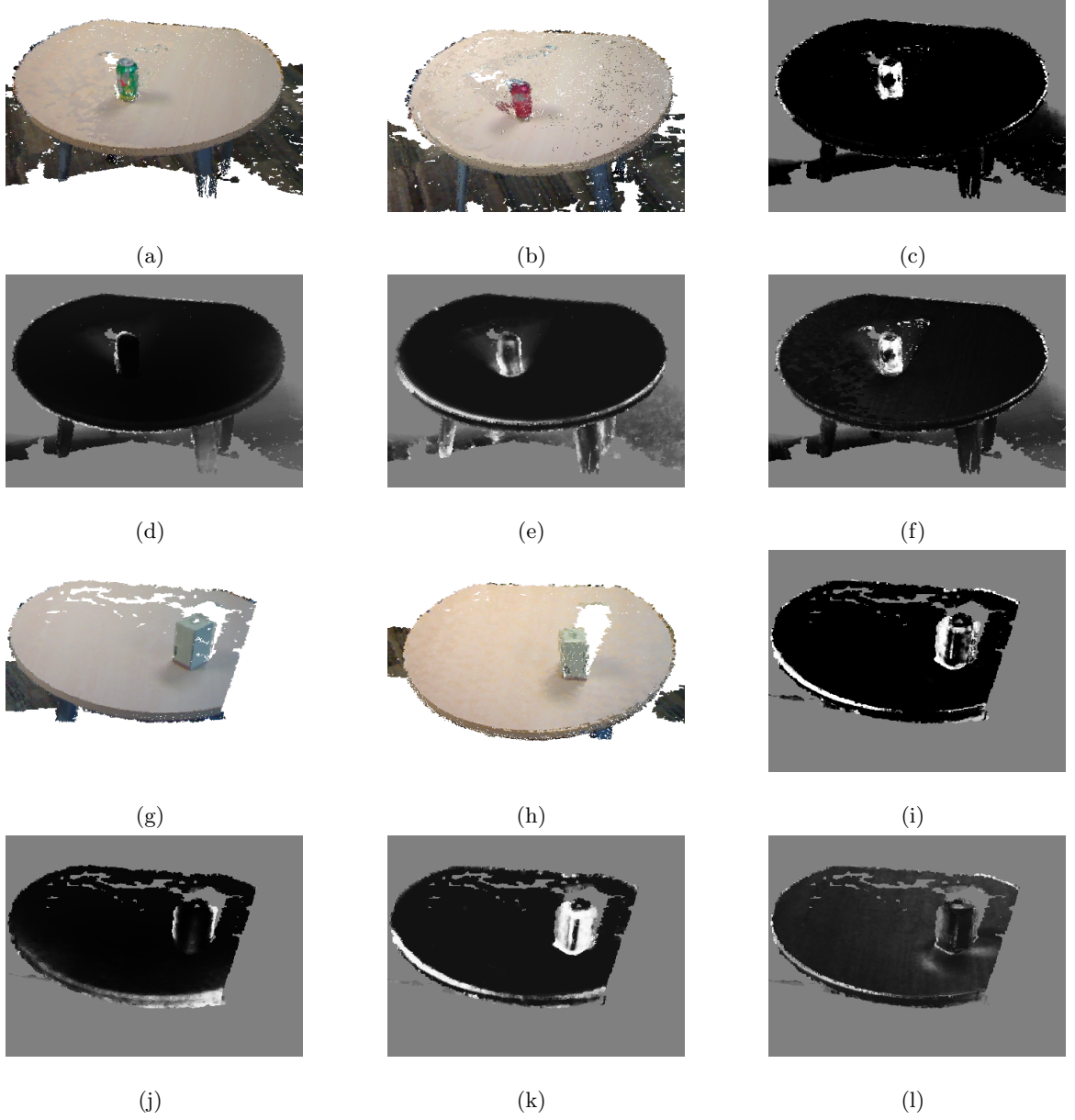


Figure 4.3: usefulness of all components of the sensor model. Two experiments in which a soda can is replaced by a different can at the same location (upper two rows), and a white box is rotated (lower two rows). The right panels in the first and third row show the motion probabilities for the left scenes using the combined sensor model given in eq. 4.3. Second and fourth row show (from left to right) motion probabilities based on depth, orientation, and color only.

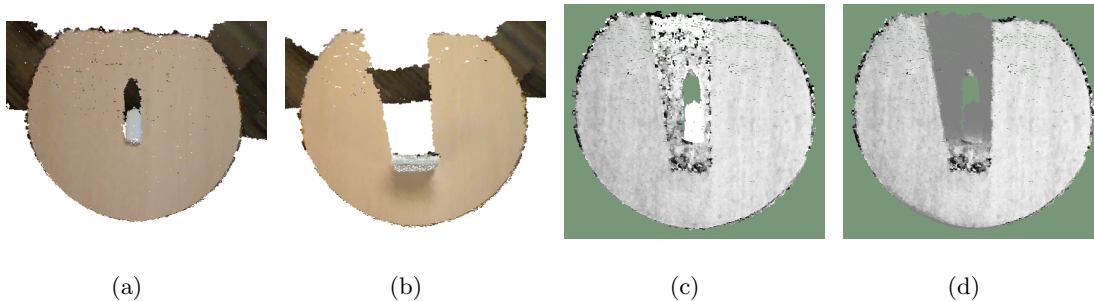


Figure 4.4: our depth-dependent color model. The white object in the left scene (a) is occluded by the white box in the second scene (b). A depth-independent model (c) generates a very high probability that the white object did *not* move. Our color model (d) correctly captures the occlusion and generates motion probabilities near 0.5 in the occluded area. In this and further figures, green represents points either not seen during mapping or seen in one map but not the other so that differencing is not possible. In (c) and (d), the floor has also been removed for clarity. Grayscale pixels give  $p(m | z)$ : the darker, the higher the probability of movement.

### *Multi-Component Sensor Model*

To evaluate the benefit of incorporating color and orientation into our sensor model, we generated two challenging scenarios in which the objects could not be detected based on depth alone. The two pairs of scenes are shown in the left two panels of the first and third row of fig. 4.3. The motion probability of the surface patches using our complete sensor model is shown in the right panels of the same rows. As can be seen, our model detects both the replaced soda can and the rotated white box. To shed light on which components of the sensor model contribute to these detections, the second and fourth row show the motion probabilities estimated from individual components only (from left to right: depth, orientation, color). The left panels in these rows show that a depth only model would not detect the object motion. As expected, the soda can is mostly detected by the color component (second row, right panel) and the orientation component is crucial to detect the rotated box (fourth row, middle panel).

### *Depth-Dependent Color Model*

Our color and orientation models take the expected color and orientation into account, but also model dependence on the expected vs. measured distance of the surface patch. In this experiment we demonstrate the advantage of our model over a sensor model that ignores this dependence (as that of [101]). The left two panels in Fig. 4.4 show top views of an experiment in which a white object is placed on the table in scene 1 (Fig. 4.4(a)) and is then occluded from the camera viewpoints by a white box in scene 2 (Fig. 4.4(b)). Fig. 4.4(c) shows the motion probabilities coming from a color model that does not model dependence on depth (a depth-independent mixture between a Gaussian and uniform color model). The model erroneously is certain that the white object in the left scene did *not* move (note the nearly black outline of the object). However, the second scene provides no evidence about this area since it is fully occluded by the white box. As can be seen in Fig. 4.4(d), our depth-dependent color model generates the correct probabilities, having near 0.5 probability in the occluded area.

### *Segmenting Moved Objects*

In Fig. 4.5 we show results on the complex scene pair of Fig. 4.1. Each scene contains a spray bottle and a cylindrical object in about the same location, such that simple nearest-neighbor point search would consider them the same objects. Due to our use of color and orientation information, we correctly detect changed surfaces at these locations in both scenes. One of the objects is also occluded, necessitating the use of our depth-dependent color model. In Fig. 4.5(a) and Fig. 4.5(c) we show the raw probabilities  $p(m | z_d, z^*)$  from our sensor model. Labels after regularization are shown in Fig. 4.5(b) for the first scene and Fig. 4.5(d) for the second scene, demonstrating that we find all movable objects with high accuracy despite using only low-level cues. We extract each connected component of same-label surface patches as a 3-D model and show the foreground (object) models in Fig. 4.7(a), along with the merge of the two background segments, which fills in some holes in each individual table.

Fig. 4.6 gives results for another scene pair. In the tvtable dataset, small objects move around on a cluttered table. In Fig. 4.7(b) we show four extracted foreground connected components along with the merged backgrounds; again we have filled in some holes on the

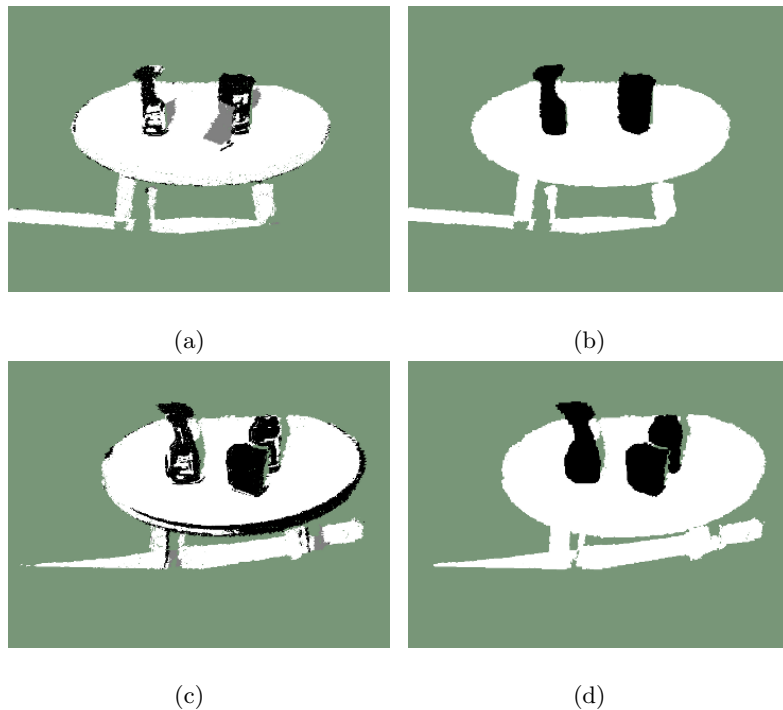


Figure 4.5: differencing with and without spatial regularization. (a) surfel-wise differencing and (b) regularization results for the first scene of fig. 4.1; (c), (d) similarly for the second scene.

table. We do not show foreground components that are very small, of which there are one in Fig. 4.5 and three in Fig. 4.6. We provide 3-D model files for these scenes, and the extracted objects, at <http://www.cs.washington.edu/robotics/projects/object-discovery/>. PLY files can be opened using Meshlab [34].

### *Quantitative Results*

We manually annotated which surfels moved for each scene pair in a set of four scenes. (Occluded points were marked not moved.) Performance numbers for these scene pairs are given in table 4.1. Precision and recall were calculated with respect to surfels annotated as moved, since those are a small minority of surfels and are the most salient surfels for most applications of differencing.



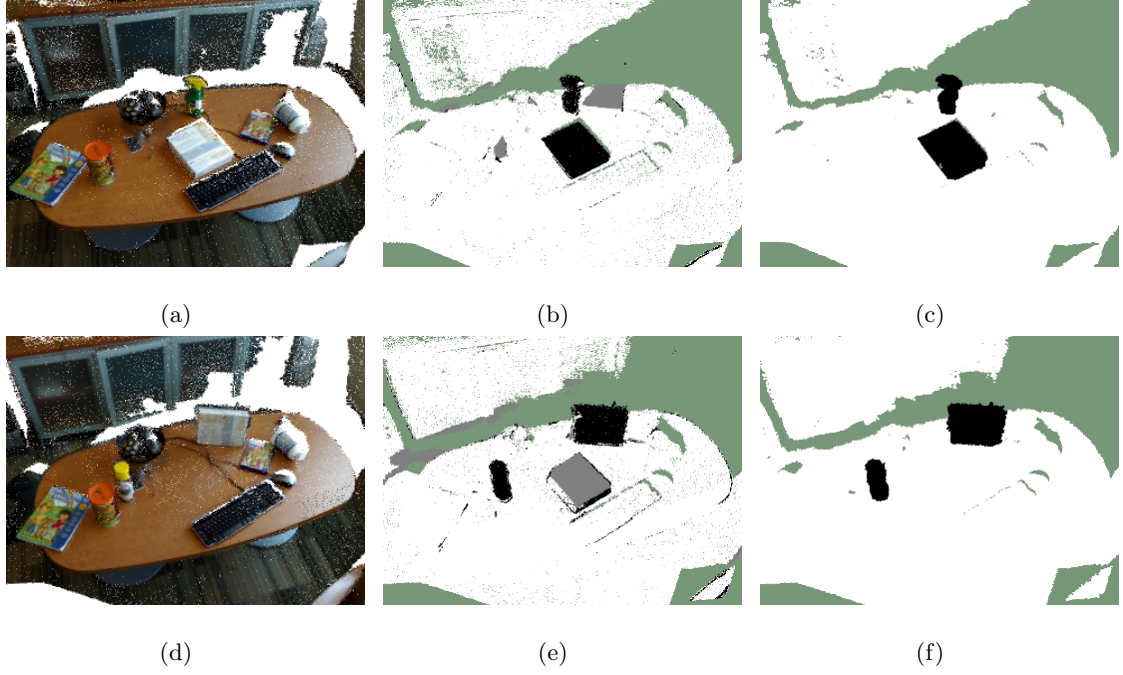


Figure 4.6: differencing with and without spatial regularization. (a), (d) two scenes from the tvtable dataset; (b), (e) surfel-wise differencing results for the two scenes; (c), (f) regularization results. A few surfels shown in the surfelwise result do not appear in the final output due to culling we perform prior to regularization. Three large occluded areas in (b) and (e) show as medium gray (movedness uncertain).

|                | precision | recall | accuracy | baseline<br>accuracy | % error<br>reduction |
|----------------|-----------|--------|----------|----------------------|----------------------|
| <b>average</b> | .965      | .800   | .990     | .968                 | 68.8                 |
| <b>min</b>     | .911      | .662   | .980     | .954                 |                      |

Table 4.1: Performance statistics aggregated over 12 scene pairs after spatial regularization. “Baseline” refers to the classifier that labels all surfels not moved. “Error reduction” is our improvement w.r.t. the baseline error rate. To pick regularization parameters we optimized a modified F-1 score giving extra importance to precision.



Figure 4.7: object modeling from differencing. Objects extracted from the scene pairs of Fig. 4.1 and Fig. 4.6, and the merged background components of each pair of scenes. In the right panel some of the background is faded for clarity.

#### 4.3.5 *Timing*

In table 4.2 we break down the time spent by our system on the scenes of Fig. 4.6 (differencing both ways). These two videos have 360 and 500 frames respectively; the maps contain 650k and 620k surfels respectively. Subsampling frames would speed up mapping, reconstruction and differencing. There are a number of other ways to speed up mapping, but in our experience they are all very data-dependent; we used settings conservative enough to work in a relatively large number of cases at the expense of speed. Alignment could be sped up by subsampling the set of features given to RANSAC; here again we are conservative. The machine used has 16 hardware threads, and various parts of our system make use of this; in particular differencing is almost embarrassingly parallel.

#### 4.3.6 *Summary and Future Work*

We have described a method to determine movable parts of a scene based on change detection (differencing) between this and another scene. Our method makes no assumption on overall object shape, texture of surfaces, or large motion relative to object size. It also models and finds occluded surfaces. Phrasing the differencing problem in terms of a probabilistic model of sensor readings provides robustness to noisy data as well as a principled

| Stage          | Time                            |
|----------------|---------------------------------|
| preprocessing  | 400s                            |
| mapping        | 600s (mostly bundle adjustment) |
| reconstruction | 360s                            |
| alignment      | 140s (100s w/o ICP)             |
| differencing   | 180s                            |
| regularization | 24s                             |

Table 4.2: Time spent in each stage of our algorithm for the scene pair shown in Fig. 4.6.

way to combine information from multiple sensors. The method relies on 3-D reconstruction, whose accuracy and robustness could use improvement; the difficulty of making good geometric maps is our largest obstacle. Our algorithm can also find large moving objects, such as furniture; see fig. 4.8 for an example.

We plan to learn some of the parameters of our measurement model, but have yet to due to time constraints. We have experimented with using the uncertainty model computed during mapping to make differencing data-dependent with respect to the expected scene, but doing this in a motivated way is extremely inefficient.

We plan to use the movement detection techniques presented here together with matching of surfaces among multiple scenes to improve object localization. With our differencing technique it is straightforward to combine evidence from multiple scenes to improve the input to spatial regularization. Matching can be initialized using point-feature matches for highly textured areas and point-cloud-based techniques for untextured regions. Then evidence about intra-scene and inter-scene similarity can be combined, perhaps using random fields similar to those of [177], to identify objects common to multiple scenes.

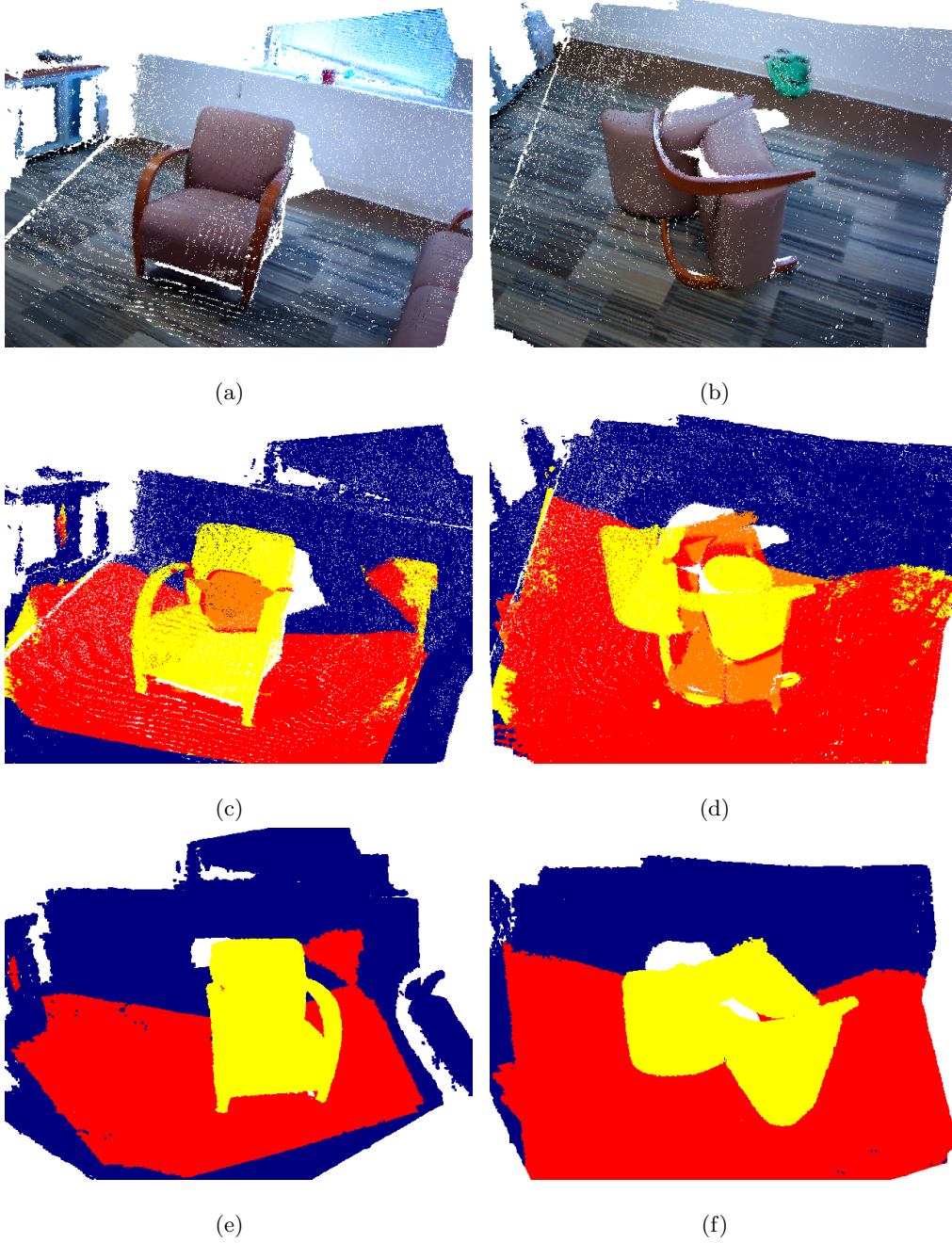


Figure 4.8: an example of using scene differencing on large objects such as furniture. (a), (b) 3-D maps of two scenes with a chair. (c), (d) results of per-point differencing of each scene against the other (aggregated over all frames). Dark blue indicates no information in the other scene at that location. (e), (f) results after spatial smoothing with MRF inference. Allowed node labels are changed (yellow), unchanged (red) and unsure (blue).

#### 4.4 Using Information from Many Scenes

The technique of sec. 4.3 can segment each of two maps of a scene. If we have several maps of the same scene, we can improve our segmentation of each of them by making use of all maps in each segmentation process.

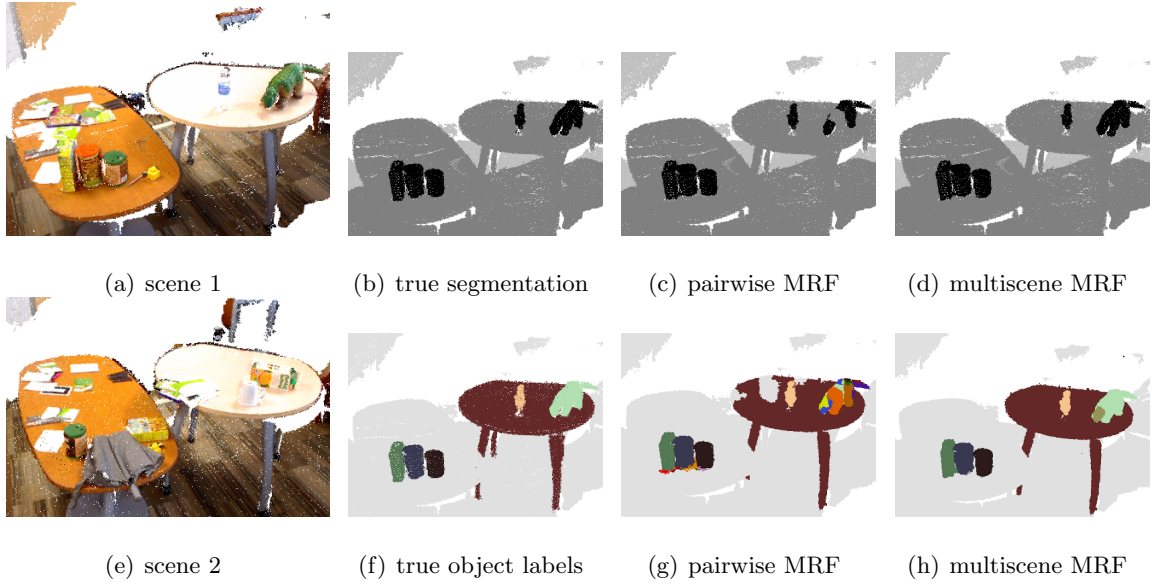


Figure 4.9: multi-scene segmentation using change detection. (a), (e) two scenes. (b) Ground truth for which surfels in the first scene move with respect to the second (black = moved; medium gray = not moved; light gray = not visible in (e)). Results of (c) pairwise and (d) multiscene differencing MRFs, showing which surfels of (a) are estimated moved/not moved in (e). The multiscene MRF uses information from a total of 10 scenes, including the two shown here. The pairwise MRF misses the shoulders of the dino (middle right of scene); the multiscene MRF does not. (f) The ground-truth object labelings for scene (a), with each object a different color. (g) Scene segmentation obtained by assigning each surfel a bit-vector label put together from the results of the two-scene MRF of sec. 4.3. (h) Scene segmentation output by our multiscene MRF, which is much more accurate than that in (g). The table is discovered as an object due to moving in one of our scenes. This figure (in particular the last row) is much better viewed in color.

In sec. 4.3 we introduced an algorithm to use RGB-D sensor readings to probabilistically decide which surfels in a scene are unchanged in a second scene. We run inference in a

Markov random field (MRF) over these probabilistic judgments to create a binary (moved vs. not moved) segmentation of each scene. We use this pairwise scene differencing algorithm to segment each visit to a scene by using change detection with respect to each other scene in the dataset.

A straightforward application of this approach to multiple scenes would be to compute a segmentation for each pair of scenes. As a result, we would get a “soup of segments” [152] for each scene, since the pairwise segmentations are not necessarily consistent, due largely to sensor noise effects differing widely between scenes. Furthermore, we would like to use our segments to reconstruct 3-D objects, so we prefer to have only one segment per object per scene to simplify the decision of which surfels to include in each reconstructed object.

To obtain a single consistent segmentation per scene, we combine information from differencing between all scene pairs into a single labeling process. Instead of labeling a surfel as 0 or 1 (moved or not moved) for each pair of scenes, we generate a joint label vector  $l \in [0, 1]^{n-1}$  for each surfel in scene  $i$ . The  $j$ th bit of this label vector represents whether the surfel changed between scenes  $i$  and  $j$ . To jointly reason about all surfel labels in a scene, we perform spatial regularization using a Markov random field (MRF). Each node corresponds to a surfel and is a random variable ranging over the values of that surfel’s label vector. Connections between neighbors increase spatial consistency of label vectors by providing soft constraints on the transitions between labels.

Our data energy for label  $l$  at surfel  $s$  of scene  $i$  is

$$D(s, l) = \sum_{j \neq i} \begin{cases} p(s \text{ moved in } j), & v_j(s) \text{ and } l_j = 0 \\ 1 - p(s \text{ moved in } j), & v_j(s) \text{ and } l_j = 1 \\ 0 & \neg v_j(s) \end{cases},$$

where  $j$  is a scene,  $l_j$  is the  $j$ th bit of  $l$ , and  $v_j(s)$  is true iff surfel  $s$  was visible in any frame observed in scene  $j$  ( $v$  stands for “visible”; surfels in  $i$  that are never seen in  $j$  contain no evidence for having moved or not, represented by a zero data energy term, or cost).  $p(s \text{ moved in } j)$  is determined by pairwise scene differencing.

The MRF smoothness energy imposed on neighboring surfels is the Potts model (energy is 1 if labels differ, 0 if not). This smoothness term on the joint label  $l$  links together all the scenes: neighboring surfels typically correspond to the same object and thus should move consistently w.r.t. to any other scene. This energy is weighted by the local 3-D curvature



of the scene to discourage segment boundaries from being placed in low-curvature areas. While such a weighting is not perfect (for example, papers on a table can move around without causing high-curvature edges), it gives good results in our test scenes.

We use graph cuts to perform MAP inference in the MRF. We then take each connected component of surfels with the same label to be a possible object. For a set of  $n$  scenes, one run of inference in this multiscene MRF makes use of the same information as  $n$  runs of inference in two-scene MRFs. An example can be found in fig. 4.9.

While in this work we use only motion as a cue for segmentation, other cues such as shape and color priors could be included in our framework. In principle, our multiscene MRF can combine any number of binary segmentations, probabilistic or not, by including them as entries in the bit vector that we use as a label.

#### 4.4.1 *Efficient Implementation*

The label set for each surfel in a scene has size exponential in the number of scenes. For 10 scenes of moderate size we are able to run inference in under four hours. However, a good way to speed inference is to reduce the label set size, as graph-cut inference (using the expansion-move algorithm of [23]) has complexity linear in the number of labels. Fortunately, the label vectors we create from pairwise differencing results provide a good estimate for which surfel moves we expect between a scene and any other. Suppose only two objects in scene 5 move, one in scenes 7 and 13 and the other in scenes 3, 7 and 10. Then for scene 5 we expect to see many label vectors with 1s in the 3rd, 7th, 10th, and/or 13th components and 0s everywhere else. Therefore we discard label vectors that occur rarely in pairwise results. Specifically, we sort label vectors of all surfels in a scene by their frequency (estimated from pairwise differencing) and only instantiate the most frequent labels such that at least 95% of surfels are represented. Fig. 8.3 shows the dramatic speed improvement this gives us. Empirically we see virtually no difference between the results with full and approximate label sets.

Scalability might still be a problem for very large scene sets. However, we believe that online methods, which update an object-based representation of the scene over time after processing each new visit, will still be possible by subsampling which scenes are considered jointly or by maintaining a canonical “background” scene that contains only static objects,

and differencing each new scene only against the current background scene.

## 4.5 Object Segmentation from Motion with Dense Feature Matching

### 4.5.1 A More General Segmentation Algorithm

There are two scenarios that cause scene differencing to fail as an object segmentation method. An object may move only slightly, so that the shape and color at most locations on its surface change very little. [72] calls this type of motion “perturbation” and analyzes it separately from large motions. Another failure case is that a set of objects touching each other move together to another configuration in which they touch. Segmentation by scene differencing requires that objects be physically separated, since there is no large-scale appearance matching involved. Therefore we now describe a feature-matching-based segmentation method, which is indifferent to the magnitude of motion. In order to avoid assuming we have very distinct sparse features, we match relatively indistinct ones, and to improve dense segmentation, we match features computed at dense rather than sparse locations. First we calculate features (we use mostly color and surface curvature histograms) at a large subset of points (for efficiency we select a subset, either randomly or by using some notion of how interesting the local shape is). We feed all point matches, with matching scores, into a sample consensus motion fitting algorithm, which was introduced in the 1980s [56] but is still the state of the art for model fitting, even when given a low percentage of correct correspondences.

### 4.5.2 Introduction

It’s reasonable for the notion of an object to depend on the capabilities of the robot, so that a robot consisting of a camera and a manipulator arm, for example, is only interested in objects it can both see and manipulate. Our robot has a hand on the end of an arm, so in this section we think about objects light enough to be moved by the hand. Once we are using movability as a measure of objectness, it is natural to use motion as a cue to find objects. Assuming the environment is also populated by humans, objects will move over time, so that the robot can leave a *scene* (local piece of the environment), return some time later, and find some objects by running a passive change detection algorithm. However, there will always be some objects that humans don’t move very often or that rarely appear



without surrounding clutter. These scenarios motivate *active object segmentation*, in which the robot literally takes matters into its own hands and causes objects to move.

For purposes of this work we assume the robot is able to push surfaces in the environment until it detects that something has moved. Active exploration for finding objects is not a solved problem; recent approaches all start by fitting a table plane or assuming the background is a single color. However, we focus on segmentation and pose estimation of the moved object(s). We use RGB-D camera frames from before and after the movement, but not during; using all frames would allow us to employ tracking methods, but would require extra work to track the robot arm (since the arm’s reported pose matches its actual configuration only roughly [109]) or the arm of the human causing the motion.

Given two views of a scene containing objects that undergo more or less arbitrary motion, some sort of local matching is required to segment objects. If all objects are highly and distinctively textured, descriptors may only have to be computed at sparse keypoints. Our objects are not all highly textured, so we choose to match descriptors at dense keypoints. One natural set of dense keypoints is the set of all samples we have of the scene, and these are the keypoints we choose to use in this work. We propose a random-sampling algorithm to enumerate possible object motions and identify those most consistent with the scene. The algorithm works very well for well-separated objects despite the high percentage of false point correspondences considered. We can use the estimated motions either to immediately produce a segmentation or to derive a measure of our uncertainty about the correct segmentation, which we can use as a termination condition in an iterative segmentation process.

#### 4.5.3 Related Work

In previous work (sec. 4.3) we introduced a change detector for RGB-D scenes given a global alignment of the scenes. It incorporates depth, color and surface orientation cues as well as modeling occlusion. As we will discuss later, from this model we can derive a geometric consistency score for a hypothesized pose of a set of surface samples with respect to another scene. We will show in sec. 8.4.2 that this score is a better measure of geometric consistency than various scores derived from ICP [15]. Here we use this *scene differencing* operation both to identify surfaces that move during a push and to score candidate motions. This

work improves on the segmentation method of sec. 4.3 by using motion fitting to help locate object boundaries. Ch. 8 describes a method for matching objects at different times once the objects have been segmented; this work instead takes the approach of segmenting each scene and matching objects across scenes simultaneously, with the happy side effect of also estimating each object’s motion, which the method of ch. 8 is not able to do.

#### 4.5.4 Motion Detection for Alignment

In ch. 3 we introduced a probabilistic measurement model for RGB-D cameras and used it to detect changes between two scenes  $\mathcal{S}_0$  and  $\mathcal{S}_1$ . In this chapter, these scenes are snapshots taken before and after a hypothetical robot attempts to manipulate the environment (e.g. by pushing a selected surface point).  $\mathcal{S}_i$  is represented as a set  $\mathcal{M}_i$  of surface samples, which might be a single RGB-D frame or an aggregated map, plus a set of camera views  $\{V_{ij}\}$ , each of which also consists of surface samples. We use a view-based formulation of change detection to account for occlusion. For each sample in  $\mathcal{M}_i$  seen from each view  $V_{(1-i)j}$  we estimate the probability that the surfaces represented by the sample and by the pixel it projects to in the view are the same. We use only very local information about the surfaces to make these judgments, in order to avoid smoothing information across object boundaries. Given a hypothesized alignment, we project each surface sample  $s \in \mathcal{M}_0$  into each view of  $\mathcal{S}_1$  to obtain the expected measurement  $z_s^*$ , which consists of depth, color and surface orientation. Denote by  $z_s$  the surface sample from  $\mathcal{S}_1$  that projects to the same point on  $V_{1j}$ ’s image plane as  $s$ . A *sensor model* provides  $p(z_s \mid z_s^*)$ . Since we use the model for change detection, we actually have two sensor models,  $p(z_s \mid m_s, z_s^*)$  and  $p(z_s \mid \neg m_s, z_s^*)$ , for the cases in which  $s$  has and has not changed. We denote the boolean variable specifying whether a surface sample has changed, or “moved”, by  $m_s$ .

For purposes of this chapter, each scene consists of a single view. We estimate the probability of change for  $s$  as

$$p(m_s \mid z_s, z_s^*) = \frac{p(m_s \mid z_s^*)p(z_s \mid m_s, z_s^*)}{p(z_s \mid z_s^*)} \quad (4.9)$$

$$\approx \frac{p(m_s)p(z_s \mid m_s, z_s^*)}{p(z_s \mid z_s^*)} \quad (4.10)$$

$$\propto p(m_s)p(z_s \mid m_s, z_s^*), \quad (4.11)$$

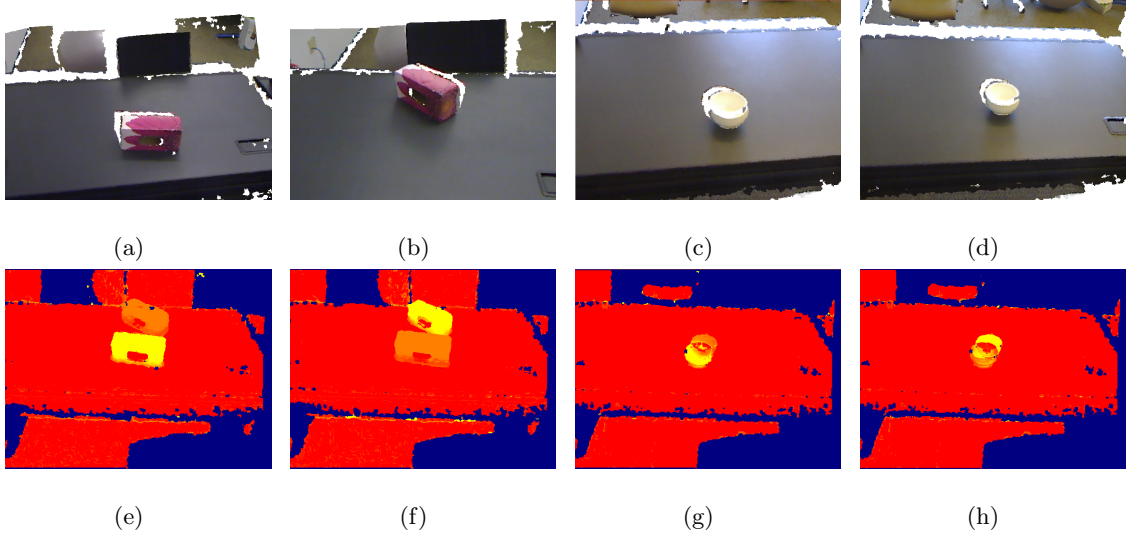


Figure 4.10: scene differencing on two before-and-after-push frame pairs. This and future figures show scenes of household objects on a table, with the floor and chairs in the background. (e), (f) differencing results for the frames of (a) and (b), in which the object moves far enough that there is no overlap between its before- and after-push 3-D positions. (g), (h) differencing results for the frames of (c) and (d), in which there is overlap. The brighter yellow the differencing result at a point, the higher the probability that that surface changed. Orange points have high uncertainty of  $m_s$ , usually because they are occluded in the other scene. Yellow points will become seeds for RANSAC.

where we can make the last approximation because we only use these formulae to maximize over  $m_s$ . The forms of the distributions we use can be found in sec. 3.1.

From this change detection model we can derive a score for a hypothesized rigid alignment. Given a set of surface samples in the first scene and a proposed alignment, we can run differencing at each point sample and aggregate some statistic over all surface samples to get a score. Here we choose to score a hypothesis  $X$  with

$$S(X) = \sum_{\text{sample } s} \max(0, 0.5 - p(m_s \mid z_s, z_s^*)). \quad (4.12)$$

The higher this score, the better the alignment. This score rewards surfaces that match well, avoids penalizing occluded surfaces, and avoids over-penalizing surfaces that match extremely badly, which is helpful when there are likely to be extraneous points among the

surface samples.

In addition to computing this geometric consistency score for a given hypothesized pose, we can optimize the score. This objective is highly nonconvex, so initialization is necessary; for this work we will initialize the optimization using the highest-scoring of a large set of poses proposed by random sampling. This optimization is similar to that of ICP, but handles color in a non-heuristic way (as opposed to, e.g., the popular methods of running ICP in a combined space of position and color and of restricting possible correspondences using color distance thresholds), does not require choosing point correspondences, and models occlusion properly. We set up the optimization as nonlinear least squares and optimize with Levenberg-Marquardt (LM). LM has a minimum required number of “measurements” (of which the objective is the sum of squares), but the more measurements are used, the slower the optimization. We optimize 6 parameters—translation, roll, pitch and yaw—so we divide the surface samples into  $N$  groups; each of  $N$  “measurements” is the sum of  $S(s)$  over the corresponding group of samples. Using each surface sample as a separate measurement might yield a better optimization objective, but with  $N = 16$  the optimization already takes multiple seconds for even moderately sized clouds (10,000 points).

#### 4.5.5 Motion Estimation

Pseudocode for our algorithm to identify the rigid motions that occur in the scene is shown in alg. 4.2. First (line 1) we run passive change detection over the before- and after-push frames; then (line 2) we identify the points that most clearly do not agree with the identity motion. These are the points whose motion we know we need to explain by proposing other motions. (There may also be points that agree well with the identity transformation but that we will want to explain otherwise.) For this reason and for efficiency, we use only these points to generate motion hypotheses to be evaluated. For this paper we define “do not agree” to mean  $p(m|z_s, z_s^*) \in [.05, .5 - \epsilon] \cup [.5 + \epsilon, 1]$ , excluding points from one scene whose hypothesized position in the other scene is occluded (in which case  $p(m|z_s, z_s^*) = .5$ ), since we have no positive evidence that those points are not present in the other scene. We use these “non-agreeing” points as *seed points* for random-sampling-based motion fitting.

We use local RGB-D descriptors to select point correspondences from which to generate rigid transforms. Over each point’s 3-D neighborhood we compute a histogram of

---

**Algorithm 4.2** our algorithm for iterative rigid motion estimation with dense descriptor matching.

---

**Input:** RGB-D frames  $F_0, F_1$

---

- 1: Difference  $F_0, F_1$  using the identity transformation
  - 2: Select seed point sets  $S_0 \subset F_0, S_1 \subset F_1$
  - 3: Compute descriptors and cross-frame descriptor similarities for seed points
  - 4: **while** there are seed points unexplained by a transformation **do**
  - 5:     Randomly generate many transformations  $X$  and score using eq. 4.12 with an approximation to [76]
  - 6:     Rank all transformations by approximate score
  - 7:     Keep only the top  $k$  transformations and rerank them using eq. 4.12 with [76]
  - 8:     **for** each  $X$  by score **do**
  - 9:         Find inlier points  $I_0 \subset F_0, I_1 \subset F_1$  for  $X$
  - 10:        Iteratively reoptimize  $X$  to fit  $I_0$  and  $I_1$  and reselect inliers
  - 11:        **if** enough inliers **then**
  - 12:            Declare a new object and remove  $I_0, I_1$  from the set of points to consider
- 

curvatures, a histogram of colors and a histogram of color gradients. If we expected relatively high texture in color or shape we might also use more distinctive descriptors such as spin images [92] or a point-cloud version of orientation-invariant SIFT. Parallelized over 16 hardware threads, this takes about 5 seconds per frame on a 2.7GHz Intel Xeon X5550. We concatenate the histograms and compute the  $L_1$  distance from each before-push seed point’s descriptor to each after-push seed point’s descriptor. On an NVIDIA GTX580 GPU, distance computation takes no more than 10 seconds when there are 30,000 seed points per frame. For memory reasons we subsample seed points in one of the two frames to 30%.

There are a variety of existing approaches to matching multiple objects undergoing rigid motion between two scenes. Gordon and Lowe [65] compute SIFT descriptors for sparse keypoints and match them with the standard ratio test, which works when there are a large number of highly distinctive neighborhoods on all objects. If there aren’t individual distinctive neighborhoods but there are clusters of descriptors for sparse keypoints that are well separated at the cluster level, the method of Collet et al. [88], which quantizes SIFT descriptors and requires only that each match be to a descriptor in the same bin, is appli-

cable. However, we want to be able to match objects like single-color dishware, for which no set of sparse keypoints will produce distinctive descriptors, assuming it’s even possible to choose sparse keypoints intelligently. Therefore our method generates and scores a large number of matches between densely sampled points that have high descriptor similarity.

To generate each motion proposal (pseudocode line 9) we choose one seed point from each frame and a rotation angle about the first seed’s normal axis. Together with a deterministically defined rotation about the second-frame seed’s normal, these uniquely identify a 3-D transformation. We score each transformation using an approximation to the score discussed in sec. 4.5.4 that uses only the depth component of the sensor model and is readily implemented on a GPU. We can run this approximate scoring on about 5000 hypotheses per second, as opposed to 75 per second with full scene differencing. In order to avoid running the full scoring operation on too many proposals, we rank proposals by approximate score and run the full model on only the top  $k = 100$  at a time. This mini-batch structuring is a compromise between generating and processing hypotheses one at a time, such that we can stop as soon as most seed points have been explained by motions, and generating and scoring all possible hypotheses in one batch, such that we know the best-fitting hypotheses possible are in our list of options.

Once we have selected a short list of hypotheses to consider further, we run scene differencing over the seed points in each frame with respect to the other scene, and compute inliers as those points with low  $p(m|z_s, z_s^*)$ . We then restrict the set of inliers to the connected component including the seed point. This step is motivated by cases in which the seeds are taken from one object but the inliers are mostly on another object, as in fig. 4.11. It’s not clear whether to approach such cases by trying to figure out that the object containing the seeds doesn’t match well or by shifting our focus to the object with most of the inliers, so we instead cull proposals whose inliers aren’t near the seeds.

Each hypothesis in turn is refit to the inlier set using the optimization discussed in sec. 4.5.4, after which (line 12) if there are enough inliers remaining we declare the inliers to constitute a new object. We then remove these inliers from consideration for future hypotheses, to counter the well-known proclivity of RANSAC methods to find only the largest consensus group. At the end of this iterative process, we have a set of motions each of which had a large number of inliers. In general there will be more candidates than true motions. For example, the inlier sets found for the frame pair of fig. 4.11 are shown

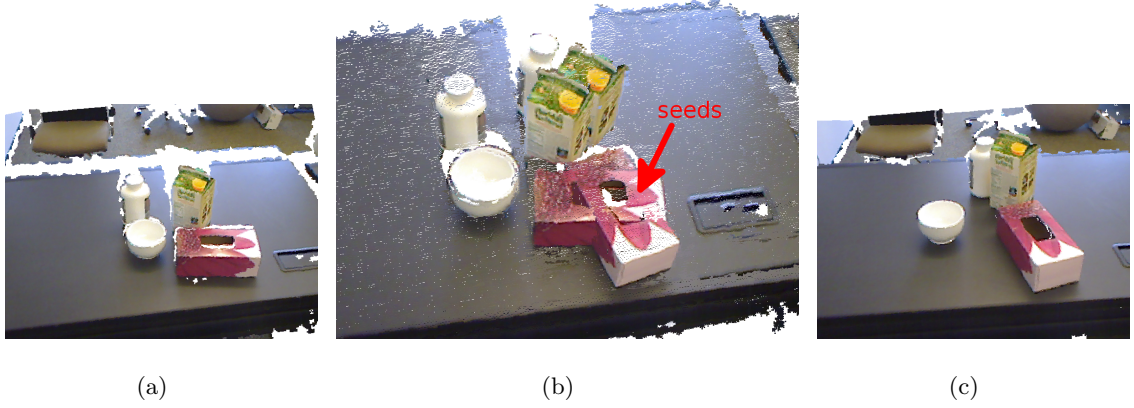


Figure 4.11: an example proposed transformation that scores well but is incorrect. This transformation scores well because of a large number of inliers on an object (the white bowl) other than the one from which the seed points for generating the transformation were taken (the tissue box). (a), (c) the before- and after-push frames; (b) the two clouds overlaid using the proposed transformation (which in this case is close to the identity). Most of the inliers for this transformation are not near the seed points used to fit the model; unfortunately, due to the texture of most of the box being the same, the box provides little evidence against this transformation even though there are likely other proposals that fit the box better.

in fig. 4.13; there are four objects and random sampling finds five motions. We therefore jointly select one motion at each surface sample in a further structured inference step.

This algorithm is very slow in general because of the quadratic number of local descriptor matches to make (we do this on the GPU but it still takes multiple seconds) and the very low inlier ratio given to the sampling-based motion fitting step.

#### 4.5.6 Segmentation

The inlier selection step of our random sampling algorithm gives us a dense assignment of motions to surface samples, so unlike [142, 57, 96, 32], we could use the result of this step as a final segmentation. However, our inlier selection is mostly random, so we postprocess the short list of candidate motions to choose one at each surface sample. Selecting the best-fitting candidate at each point is an improvement over using the inlier sets to provide a segmentation, but we can do better with joint inference. For each surface sample in each

frame (before and after) we create a node in a Markov random field, with a discrete variable specifying which candidate motion is selected at that point. We use Potts potentials for binary cliques between spatially neighboring nodes: penalize if the two motions selected are different.

In sec. 4.5.7 we show examples of using MAP inference in such MRFs to provide an immediate segmentation of a scene after a single push, but more importantly also of using marginal inference to define a measure of local object-boundariness, or segmentation uncertainty, at each surface point. We plan to use this segmentation uncertainty to decide when to terminate an iterative active segmentation approach to understand all objects in a scene.

#### 4.5.7 Experiments

First we show an example of segmentation of a simple scene with a single moving nontextured object, in fig. 4.12. The motion is small enough that the initial and final object positions overlap, so differencing by itself doesn't identify the whole object. Our full method, however, is able to identify the entire object both before and after the motion.

We have also run the method on datasets containing multiple moving objects, such as that shown in fig. 4.13, which contains four objects close enough together to present problems to state-of-the-art static RGB-D segmentation algorithms. We find five motions, although the inlier sets for the motions don't correspond to the same objects in the before and after frames. In particular, in the before frame the white bottle is included with the inlier set of the white bowl. To avoid this we probably want to replace the connected-components step in our inlier finding procedure with something with fewer hard thresholds. For this dataset we also show motion-derived boundariness. The boundariness measure varies much less with MRF parameters than the MAP segmentation does. Since the uncertainty of the segmentation of the scene is what we really want for use in future processing, this robustness is good.

Finally, fig. 4.14 shows a segmentation and boundariness for a scene with multiple objects only one of which moves. This is the ideal case for our future robot experiments: the robot manages to successfully push only one object, and pushes it far enough that its initial and final positions don't overlap, so that ideally the object can simply be picked out of the differencing results. In reality, we identify multiple motions and the regularization step fails



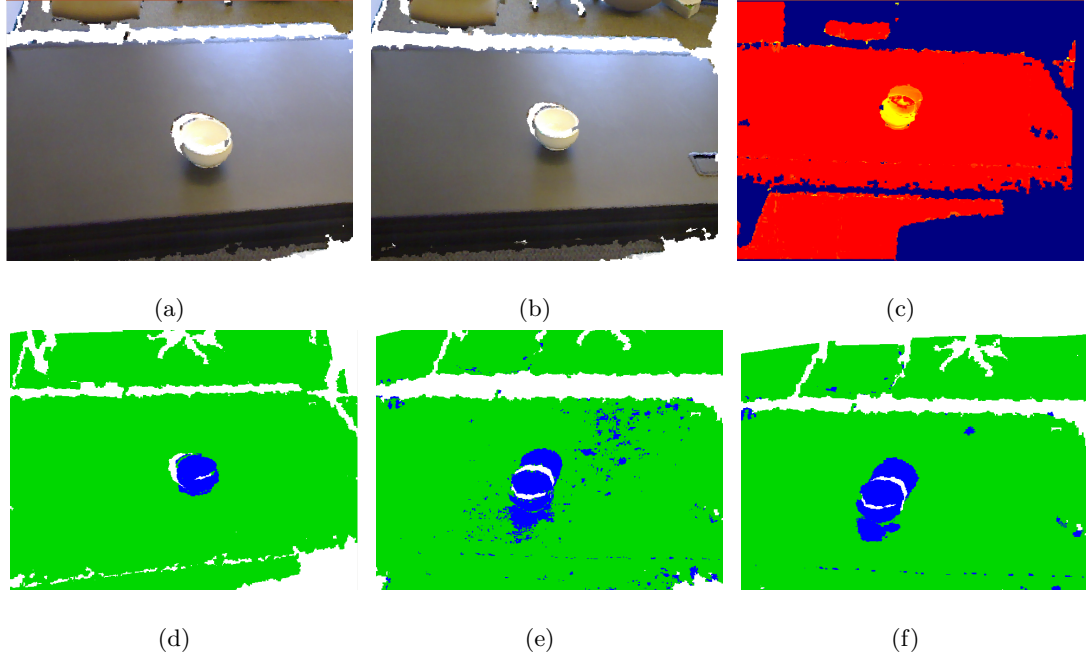


Figure 4.12: segmentation results for the bowlMediumMotion dataset. (a), (b) the before- and after-push frames. (c) the result of change detection over (a) with the no-motion hypothesis; the brighter yellow the points, the more likely the model thinks they moved. (d) inliers in frame (a) for the different motions we identify (identity transformation in green). Segmentations from MRF inference with (e) unary potentials only, and with (f) unary and spatial smoothness potentials.

to completely remove them.

Fig. 4.15 compares, on the dataset of fig. 4.13, the result of this method with that of a baseline approach based on our previous work on change detection and object discovery. In the baseline approach, we use change detection on each frame with respect to the other to label each 3-D point as changed or not. We connect points in each frame into a graph using local point distribution information, take each connected component of points marked changed to be a separate object (shown here in different colors), and solve the assignment problem [131] over these two sets of segments, using the change-detection-based segment matching method of 8.3.3 to score pairs of segments. This baseline method does poorly due mostly to the fact that the objects in this pair of scenes don't move entirely out of the space they previously occupied; this case is hard for change detection. Another factor is the

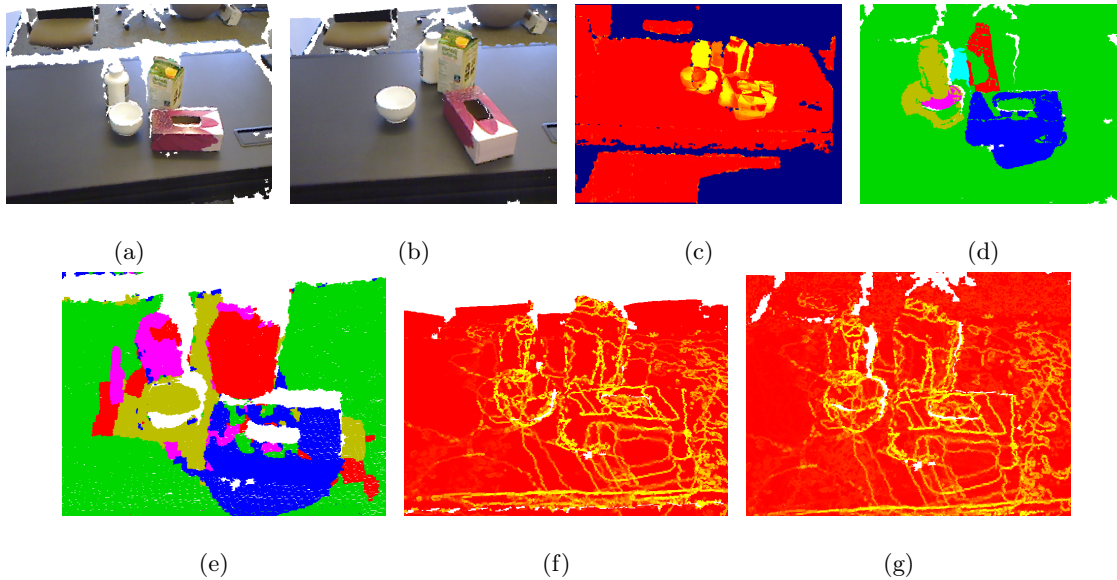


Figure 4.13: segmentation results for the multiobjWellSeparated1 dataset. (a), (b) the before- and after-push frames. (c) the result of change detection over (a) with the no-motion hypothesis; the brighter yellow the points, the more likely the model thinks they moved. (d) inliers in frame (a) for the different motions we identify (background motion in green). (e) segmentation from MRF MAP inference. (g) probability-of-boundary map computed from MRF inference (brighter = higher probability). Subfigures aren't pixel-aligned; these are views of point clouds.

need to make hard decisions on segment boundaries before the segment matching step. By contrast, the dense-feature-matching-based method is better able to identify the gross shape of the objects, and some of the correspondences between segments over time, although the fine outline of segments suffers.

#### 4.5.8 Summary and Future Work

We demonstrate segmentation and registration of multiple objects undergoing 3-D rigid motion, without relying on table plane fitting or distinctive keypoints. The method makes heavy use of a change detection model that operates on surface samples. It typically runs in 1 - 4 minutes on an 8-core 2.7GHz CPU when each scene has around 250,000 points, and is not fully optimized, leading us to believe that matching of dense point sets can be accomplished efficiently on recent hardware, one of our goals in this project.

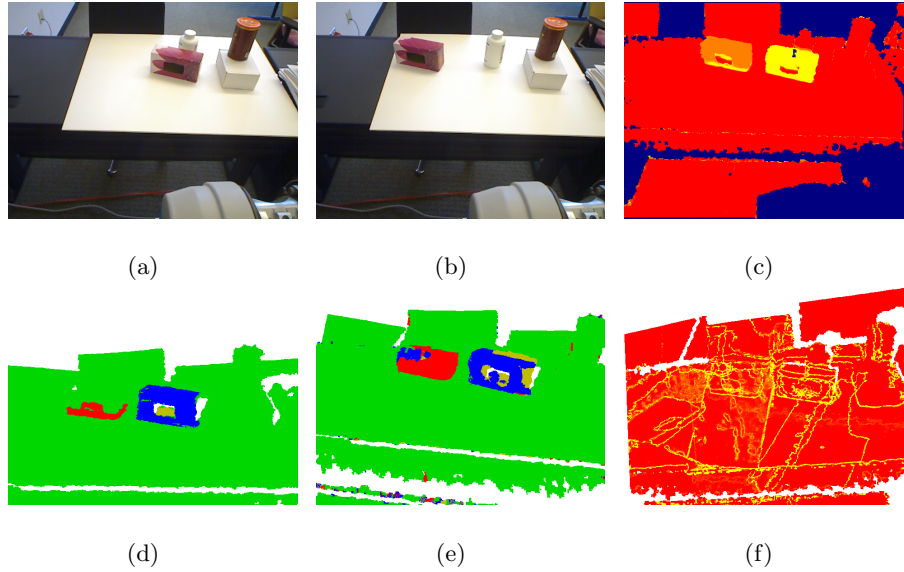


Figure 4.14: results for a dataset with multiple objects only one of which moves. (a), (b) the before- and after-push frames. (d) inliers in frame (a) for the different motions we identify (identity transformation in green). (e) segmentation from MRF inference. (c) the result of change detection over (a) with the no-motion hypothesis. The brighter yellow the points, the more likely the model thinks they moved. (f) the probability-of-boundary map (yellower = higher probability). The thing to notice is that the brightest lines include the edges of the object that moved. It’s not clear to us yet how to use this information, but there is clearly signal here.

Currently objects usually have to be well separated in 3-D for us to find them. This is mainly because of our inlier selection method; the use of connected components is reasonable when objects are well separated, but it’s less clear whether it will work for objects touching each other. Clearly the current method could not handle the scenario of a kitchen crowded by stacks of piled dishes. To approach such hard scenes we could try, rather than a RANSAC-like procedure to select a small number of proposed motions, a (much more expensive) clustering of points according to their probability of change under each of a large set of motions.

One insight into dealing with crowded scenes comes from Hager and Wegbreit [72], who point out that objects can be processed in order of occlusion. In our context this means that we can iteratively focus on explaining some set of points that are not occluded by

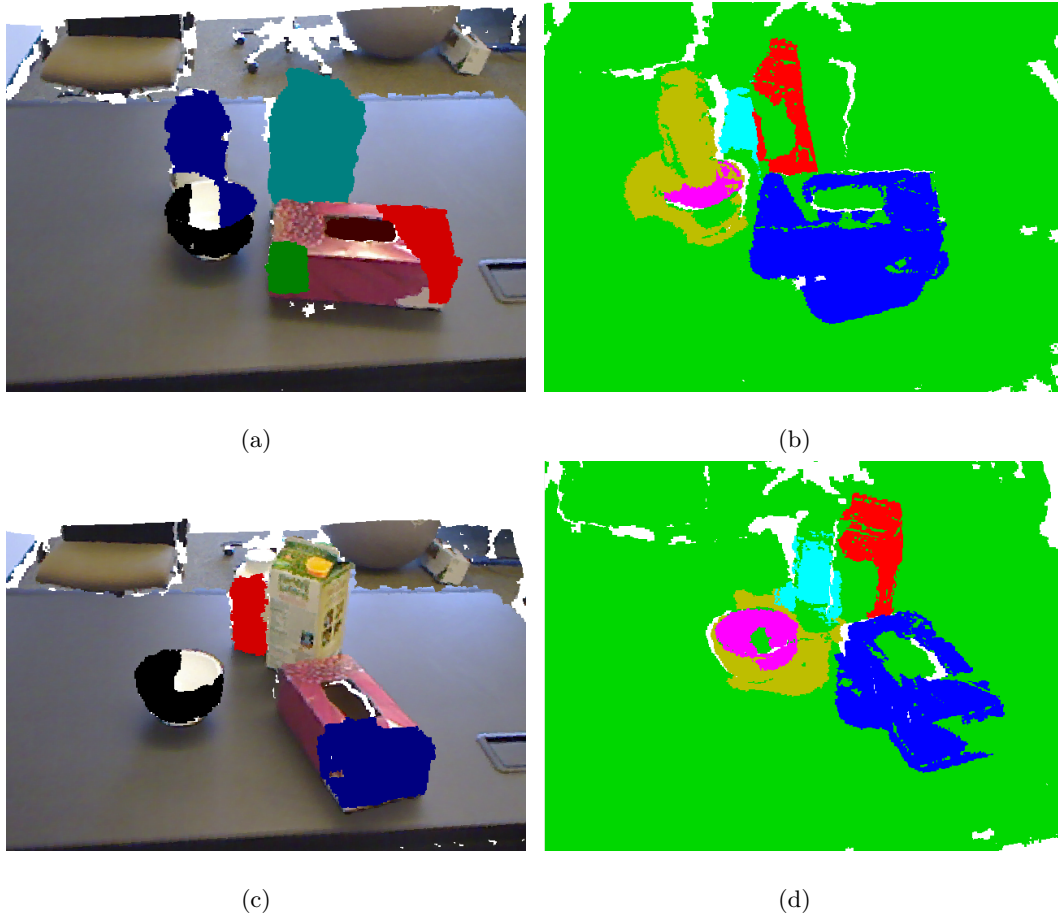


Figure 4.15: comparison of segmentation and matching results to a method based on methods from sec. 4.3 and ch. 8. Results are shown for the two RGB-D frames of figs. 4.134.13(a) and 4.134.13(b). (a), (c) results of the baseline method, described in the text. Segments in the two images that have the same color were determined to be the same object. (b), (d) results of the method of this section. Again, the same color in both images refers to the same object. Here the background is green (these points were determined to have a separate rigid motion, which in this dataset is the identity, although that is not assumed).

any others. At any point in time, most of the points in the scene are not very important to us; the challenge in using this insight is in figuring out which points we don't need to understand well yet.

## Chapter 5

# OBJECT SEGMENTATION IN DYNAMIC SCENES

### 5.1 Introduction

The segmentation techniques of ch. 4 are designed to work on sets of still images or pre-built maps. They cannot analyze motion that occurs while the robot is viewing a scene. To be able to make use of information from such situations we have to be able to perform segmentation using short-term, small-magnitude motion, as well as using video containing people who are moving the objects we are interested in.

### 5.2 Related Work

There is a large amount of prior work on motion segmentation from video; here are a few examples. Brox and Malik [26] cluster points through an entire video using pairwise similarity measure over point trajectories to perform motion segmentation of a sparse point set. The particle video technique [155], a dense motion estimation technique frequently used in an out-of-the-box fashion, tracks a semi-dense set of locations throughout a video using optical flow augmented with sparse feature tracking. Given a noisy mapping from scene elements (e.g. pixels in 2-D or surfels [139] in 3-D) to vectors representing motion between two time points, several authors have clustered motion vectors in various ways to segment one or more objects from a static background. Pundlik and Birchfield [142] track visual point features, and allow tracks to move between clusters after each new video frame. Unlike the work we present here, none of these methods produces dense segmentations.

As in ch. 4, we here describe algorithms for segmentation using motion cues with as little human involvement as possible. First we discuss an algorithm using dense motion estimation for segmentation from a video, rather than a small number of still images, of moving objects. We then describe a method for using a semi-structured human interaction with an object to provide segmentation information to an observing robot. The interaction required is limited to picking up and putting down an object, with some restrictions on the motion. None of the algorithms discussed in this chapter or ch. 4 dominates any other; each

is most useful in different situations.

### 5.3 RGB-D Flow: Dense 3-D Motion Estimation Using Color and Depth

#### 5.3.1 Improving Segmentation Robustness

Dense-feature-matching-based object segmentation can find very general motions, but relies on data-independent pre-filtering steps to limit the number of data processed at each step, which is not robust. Another approach to segmentation that works in significantly fewer situations but is much faster is to use *scene flow* to estimate motion at least roughly. One way to compare these two approaches is to consider the set of motions we allow ourselves to choose from at each point. In the sample consensus motion fitting algorithm we allow a very large number of points in one scene to correspond to a particular point in the other, which means there is a very large number of possible estimated motions to analyze, so that we need a very large number of sample consensus iterations to find good motions. When to stop running iterations and accept the best model found so far is a major problem in sample consensus algorithms when given a low percentage of true inliers. Alternatively, we can estimate a single motion field over all points in the scene at once. This has the disadvantages that it will necessarily incorporate more spatial smoothing than a point-based method, because it must include spatial regularization at every step in order to produce estimates at all points, and that due to lack of local evidence and oversmoothing, some points will have very inaccurate motion estimates. It has, however, the advantage of producing only a single motion estimate at each point; if we take these estimates as truth and imagine that at least a large minority of them are close to correct, we don't need anywhere near as many iterations of sample-consensus motion fitting as we did in the algorithm of sec. 4.5. This greatly improves speed compared to using a very large number of iterations, and robustness compared to using a small number of iterations, of the previous method.

The scene flow problem is the estimation of the 3-D translation of each point in a scene between two points in time. Most commonly it is calculated from two sensor readings (here two RGB-D frames); it can also be calculated from three or more frames in temporal sequence in order to provide extra smoothing. This does not immediately provide estimated rigid motions for pieces of the scene—those are more structured than translations—but it

does give a full parameterization of the motion of the set of points in the scene, just as much as selecting point correspondences for all points does, so rigid motions can be fit relatively easily using, as before, a sample consensus algorithm. The motion fitting algorithm we use is optimized to know that there is at most one point correspondence suggested per point in either scene.

Scene flow is usually calculated over short time spans with gradient descent algorithms. To handle large motions and to speed the solution of the linear systems that are usually used in variational optical flow and scene flow methods [24, 28, 66] such as ours, it is common to compute flow at each level of a scale-space pyramid [192] in turn. Due to down- and upsampling of the flow field in 2-D image space, scale-space methods incorporate the assumption that all parts of the scene are nearby in 3-D, which is clearly untrue of most scenes. Many researchers have introduced anisotropic regularizers to make use of information about likely motion boundaries, regions in which the motion changes abruptly across pixels [132, 184, 186]. We use an anisotropic regularizer incorporating all the available static cues for object boundaries, but it is still the case that scene flow algorithms all assume small motion and relative smoothness of motion across the image. Therefore we compute scene flow only between frames relatively close in time within our videos. This has the benefit, however, of allowing us to estimate a scene segmentation at any time during an episode of motion, so that we can cease segmentation of a scene at any time if we think we have solved it.

Our scene-flow-based segmentation algorithm has more speed/accuracy tradeoffs than either of those introduced in ch. 4, as we can adjust several parameters related to the precision to which we solve linear systems, the number of linearizations we perform, and the overall level of smoothing required. This attribute is useful for building a segmentation algorithm on top of flow, because flow estimates only need to be approximately correct to be fed into a robust motion fitting algorithm that has built-in noise tolerance.

### 5.3.2 Introduction

The ability to detect and estimate fine-grained motion is a fundamental capability for many robotic applications, including person and object tracking, gesture recognition, and object manipulation. The difficulty of motion estimation strongly depends on the assumptions

made on scenes and object movements. For instance, if objects of interest can be segmented from the scene easily at all times, tracking can be performed by associating segments in one frame with segments in the next. This has led to robust techniques for problems such as person tracking [162] and manipulator tracking [110]. Assuming that the number of rigid objects moving through the scene is small and known, moving objects can be segmented and tracked via scene differencing and matching, as has been demonstrated for 2-D laser scans [177].

For more complex motion scenarios, robotics researchers often resort to feature-based approaches, where either distinctive markers are placed on the moving objects or it is assumed that the moving objects have color features that can be tracked over time. Recently, such approaches have yielded impressive results on problems such as learning kinematic models of articulated objects [168, 96]. However, their dependence on point features limits the applicability of these techniques to textured objects only.

In this section we investigate motion estimation in its most general form: we make no assumption about the number of moving objects, whether they are rigid, whether they have visual texture, or whether the camera moves. We introduce RGB-D FLOW, a scene flow algorithm that combines dense depth and color information to estimate the 3-D motion of each point in an RGB-D frame. Many existing RGB-D flow algorithms make simplifying assumptions, e.g. using point features [116, 143] or linearized flow constraints [66]. RGB-D FLOW builds on successful optical flow algorithms that use robust norms, nonlinearized data constraints and variational solutions, e.g. [24]. We extend recent optical flow energy formulations to produce dense 3-D flow.

We show motion estimation results on a variety of scenes containing human hands and bodies, robot arms and multiple moving objects. Our algorithm computes dense motion accurately and robustly, handling both small and large motions and overcoming sensor noise. We can compute fine-grained motion on small structures such as fingers noticeably better than previous work. We also apply dense flow to rigid motion segmentation, where we use flow to segment multiple objects in an active vision scenario in which a robot pushes surfaces to discover objects.

This section is organized as follows. In sec. 5.3.4 we introduce optical flow, scene flow and our objective function. We discuss using scene flow to perform motion segmentation in sec. 5.3.5. Sec. 5.3.6 shows results for flow estimation and segmentation, and we conclude



in sec. 5.3.7.

### 5.3.3 Related Work

The most common types of dense motion estimation in computer vision are optical flow and scene flow. *Optical flow* is apparent motion (translation) in the image plane; *scene flow* is translation in 3-D. Both are usually estimated as vector fields on the pixel grid. Optical flow is an extensively studied problem that goes back to Horn and Schunck [85] (global solution) and Lucas and Kanade [123] (local solution). One development of note is that of Brox et al. [24], which introduces a variational optimization in which approximations commonly made in flow estimation are pushed to an inner loop to improve accuracy. This optimization framework is now common in optical flow work. Another is that of the “combined local and global” approach of Bruhn et al. [28], which combined local optical flow techniques with global ones. Brox and Malik [25] incorporate local descriptor matching into optical flow estimation to improve results of a differential optical flow approach (in which a small-motion assumption is made) for large motions. The basic assumption of optical flow estimation is *brightness constancy*, that the color of a point is constant over time. We will extend variational optical flow to the RGB-D case, using both color and depth reasoning.

The term “scene flow” was introduced by Vedula et al. [179], who estimate 3-D flow from color and depth data by first estimating optical flow, then using a first-order approximation of the depth map. Spies et al. [165] introduce a depth analog to brightness constancy that they called the range flow constraint. Other notable scene flow techniques are that of Huguet and Devernay [90], who add explicit occlusion reasoning for pixels, and that of Vogel et al. [181], who add a local rigidity term. These papers computed scene flow from stereo pairs, from which only sparse depth measurements can be extracted with any accuracy. There are few examples of dense depth flow before the arrival of the Kinect, one example being Lukins and Fisher [124], who add color to depth flow in the Lucas-Kanade framework.

With the introduction of dense depth sensors, there has been growing interest in computing flow combining color and depth. Letouzey et al. [116] regularize flow computation using the Laplacian of the graph of points in 3-D rather than the image Laplacian. Their method is designed for large motions and, due to their choice of Laplacian, does not deal well with objects close in space. Quiroga et al. [143] track regions in color and depth, essentially

extending the KLT feature tracker [173] to 3-D. This has the same shortcomings that the Lucas-Kanade optical flow approach does. Hadfield and Bowden [70] use particle filtering in 3-D to estimate the surface in each frame. Sophisticated resampling is necessary to assure that the surface is densely represented at all times. Gottfried et al. [66] use a variational approach with 2-D regularization. Major differences are that we use robust convex norms, avoid early linearization, and carry out more evaluation. The work of Wedel et al. [183] is the most similar to ours, differing mainly in the regularization and in the dependence of various weights on depth values.

Motion segmentation in point clouds is harder than in color images alone due to irregular sampling and the high noise levels of depth sensors. One recent approach is that of Van de Ven et al. [177], who run an expensive belief propagation algorithm on a conditional random field to perform data association, motion estimation and segmentation. The approach, unfortunately, is extremely computationally expensive, partly due to the need to consider a large number of correspondences for each point, and would not scale to datasets of the size we consider.

#### 5.3.4 Dense Motion Estimation

We consider estimating the 3-D motion of points in the scene using two RGB-D frames. We want to be able to estimate motion for untextured surfaces. Optical flow and scene flow are the most common formulations. *Optical flow* estimates translation in the image plane: each pixel location  $\vec{x} = (x, y)$  in the domain of the image  $I$  is assigned a motion vector  $\vec{u} = (u, v)$  in the plane.  $u$  and  $v$  are measured in pixels. *Scene flow* estimates translation in 3-D at each pixel: the scene flow maps  $\vec{x} \in I$  to  $\vec{u}(\vec{x}) = (u, v, w) \in \mathbb{R}^3$ .  $u$  and  $v$ , parallel to the image plane, are measured in pixels;  $w$ , orthogonal to the image plane, is measured in meters. As physical motion is not measured in pixels, it would also be possible to optimize  $u$  and  $v$  in meters. However, that would make our estimates of  $u$  and  $v$  unreliable in cases of unknown scene depth, as motion parallel to the image plane is computed using a point estimate of depth at each pixel, and these point estimates are unreliable at pixels with unknown true depth.

The scene flow formulation is attractive in many ways: (1) it does not make any assumption about the scene motion, making it capable of handling moving cameras, multiple

moving objects as well as non-rigid deformations; (2) it does not rely on the availability of identifiable sparse features such as SIFT; (3) it generates motion at all the points in the scene, which is useful for motion segmentation and dense tracking tasks. While differential formulations of scene/optical flow require small motion, reasonably large motions can be recovered through the use of a scale-space pyramid, computing flow vectors from coarse to fine, and warping images across levels.

Our flow formulation is based largely on that of Brox et al. [24]. This is a *variational* method, which optimizes a global energy functional  $E$  of the flow field:

$$\vec{u}^* = \operatorname{argmin}_{\vec{u}} E(\vec{u}) = \operatorname{argmin}_{\vec{u}} \int_{\vec{x} \in I} F(\vec{x}, \vec{u}(\vec{x})) d\vec{x}. \quad (5.1)$$

Usually  $F$  is composed of *data terms*, which match image data between two or more images  $I_i$ , and *smoothness terms*, which regularize the problem. We include data terms for color and depth as well as smoothness terms.

### *Color Consistency*

All optical flow methods to date try to enforce *brightness constancy*, the idea that the color of an object is constant over time. For a pair of images  $I_1$  and  $I_2$ , this can be written as

$$\Delta_t I(\vec{x}) = I_2(\vec{x} + \vec{u}(\vec{x})) - I_1(\vec{x}) = 0 \quad (5.2)$$

Usually brightness constancy is optimized in its linearized form by approximating with a first-order Taylor expansion to make the constraint linear in  $\vec{u}$ :

$$\Delta_t I \approx I_2 + \nabla I_2 \cdot \vec{u} - I_1. \quad (5.3)$$

Eqn. 5.3 is also called the *optical flow constraint equation* (OFCE). The experience of the optical flow community has been that nonlinearized brightness constancy, as used by [24], leads to more accurate flow than a linearized version. Therefore we use a nonlinearized color consistency term.

The classical formulation of optical flow would use a quadratic penalty ( $L_2$  norm) on the equation above, as adopted in a recent adaptation of optical flow to RGB-D data [66]. In our approach, we make use of the recent experience of optical flow researchers to design our data term. First, instead of the  $L_2$  norm, we use a robust data term that uses the

Charbonnier penalty

$$\psi(a^2) = \sqrt{a^2 + \epsilon^2}, \quad (5.4)$$

a convex approximation of the  $L_1$  norm. This deals with outliers much better than does the quadratic penalty. Second, we use the constancy of both brightness and gradients, as gradients are often more robust than brightness values under real-world illumination. Our color data term is then

$$E_C(\vec{u}) = \psi((\Delta_t I)^2). \quad (5.5)$$

### *Depth Consistency*

The data term for depth looks slightly different from that for color because the measured depth of a physical point is not constant over time. The constraint we want to enforce, therefore, is that change in depth is consistent with observed depths in both frames. Spies et al. [165] propose an analog of the OFCE that they call the *range flow constraint equation* (RFCE), given two depth maps  $Z_1, Z_2$  defined over the same domains as  $I_1, I_2$ :

$$Z_2(\vec{x}) + \nabla Z_2^T \vec{u}(\vec{x}) - Z_1(\vec{x}) = 0. \quad (5.6)$$

We instead use a nonlinearized depth consistency term similar to our color consistency energy:

$$\Delta_t Z = Z_2(\vec{x} + \vec{u}(\vec{x})) - Z_1(\vec{x}), \quad (5.7)$$

$$E_Z(\vec{u}) = \mu(\vec{x}) \psi((\Delta_t Z - w(\vec{x}))^2). \quad (5.8)$$

The difference between eqn. 5.5 and eqn. 5.8 is clear: by definition, when a point  $\vec{x}$  moves in 3-D, the depth motion  $w$  is the change of its measured depth. As in the color case, we use the robust Charbonnier penalty.

$\mu(\vec{x})$  weights the depth data term against the color data term. We use a measure of uncertainty in measured depth, which grows with distance from the camera. Specifically,  $\mu(\vec{x}) = \frac{\sigma_Z(1)}{\sigma_Z(d)}$ , where  $\sigma_Z(d)$  is the depth resolution of the stereo sensor around depth  $d$  (see eqn. 3.13). This is approximately quadratic in  $d$ .

### Regularization

Regularization is necessary for variational flow computation. The flow problem is under-constrained locally; this is commonly known as the *aperture problem*. The optical flow constraint in eqn. 5.5 has no information on how a point moves perpendicular to the  $\nabla I$  direction. With regularization, flow algorithms are often capable of producing accurate motion estimates even in regions of uniform appearance where no local signal is present. The regularization term is traditionally quadratic, but a quadratic penalty on the smoothness term leads to many small changes in flow at motion boundaries rather than few large changes, so some recent methods replace this also with a robust penalty.

The most commonly used regularization is an isotropic regularizer that penalizes a high gradient in the flow [24]. Again using the Charbonnier penalty, we can define an isotropic regularization term for RGB-D data as

$$E_S(\vec{u}) = \psi(|\nabla u|^2 + |\nabla v|^2 + \beta(\vec{x})|\nabla w|^2), \quad (5.9)$$

where we adapt the usual optical flow smoothness term to the scene flow case by scaling the depth term to trade off meters against pixels:  $\beta(\vec{x}) = f^2$ ,  $f$  the focal length of the color camera, into whose coordinate frame we assume the depth maps have been put.

When optimizing eqn. 5.9, we will use the gradient of the flow field only via the Laplacian  $L$  of the flow field. This is traditionally calculated using an image Laplacian that weights all pixels equally. While an isotropic regularizer can work well for the optical flow case, we find that it's beneficial to use an anisotropic smoothing that takes into account the local gradient signals: image gradient, depth difference, and surface orientation difference. For the case of color-only data, one commonly used anisotropic regularizer is that of Nagel and Enkelmann [132], which reduces smoothing across edges. The RGB-D flow method of Letouzey et al. [116] substitutes for  $L$  a *graph* Laplacian over the point cloud of the scene, with weights decreasing with increase in depth difference:

$$l(\vec{x}, \vec{y}) = e^{-c(z(\vec{x}) - z(\vec{y}))^2}, \quad \vec{y} \in N(\vec{x}), \quad (5.10)$$

where  $N(\vec{x})$  is the 3-D neighbor points of  $\vec{x}$ . (See [116] for how to build  $L$  from  $l$ , which is there called  $w$ .) Both these methods optimize an objective including the quadratic smoothness term  $E_S(\vec{u}) = \vec{u}^T L \vec{u}$ , where  $\vec{u}$  stands for the concatenation of the flows  $\vec{u}(\vec{x})$  for all pixels  $\vec{x}$ . While our objective takes a different form (we use the Laplacian in the style of [28]

during optimization), we also use the flow gradient only via the Laplacian. One regularizer with which we experiment takes a form similar to Letouzey’s, making use also of color and surface normal. The Laplacian matrix for this regularizer is defined using

$$l(\vec{x}, \vec{y}) = 1 - \max \begin{pmatrix} e^{-c_z |z(\vec{x}) - z(\vec{y})|^2} \\ e^{-c_c |rgb(\vec{x}) - rgb(\vec{y})|^2} \\ e^{-c_n |1 - n(\vec{x}) \cdot n(\vec{y})|^2} \end{pmatrix} \quad (5.11)$$

where  $z$  is the depth,  $rgb$  the RGB color value, and  $n$  the surface normal. Fig. 5.1 shows the three components of this function for the frame pair shown in the first column of fig. 5.2. This term, inspired by Pb [127], computes an approximate probability of an object boundary between a pair of adjacent points to avoid smoothing across boundaries. Other sources of information for such a regularizer could be an improved Pb, perhaps trained on environment-specific data, or motion segmentation results from past video frames.

Due to the max operator, this regularizer likes to smooth between points unless there is overwhelming evidence that they represent different objects. A more correct formulation would use depth-dependent color and surface normal terms in the style of sec. 4.3; we have used the simpler formulation here for expository purposes.

Regularization that downweights the connection between flow at different points can leave some points that have no local signal with no sources of information if they also have weak connections to their neighbors. Therefore we include a small zero-motion bias in our objective. We use a quadratic penalty in order to penalize large flows while penalizing small ones as little as possible, because empirically we find that when we use robust data and smoothness penalties, only large flow values indicate such underconstrained points. The full objective is then

$$E(\vec{u}) = \int_{\vec{x} \in I} (E_C(\vec{u}) + E_Z(\vec{u}) + \alpha E_S(\vec{u}) + \gamma E_B(\vec{u})) d\vec{x}, \quad (5.12)$$

where  $E_B(\vec{u}) = |\langle 1, 1, \eta(\vec{x}) \rangle \cdot \vec{u}|^2$  is the flow magnitude penalty, with  $\eta(\vec{x}) = \frac{f}{z}$  balancing pixels against meters. We have abused notation to save space: inside the integral  $\vec{u}$  is actually  $\vec{u}(\vec{x})$ .

### *Variational Solution of RGB-D Flow*

Our solution of RGB-D Flow is largely motivated by and adapted from proven variational optical flow techniques such as those of Brox et al. [24]. We have three Euler-Lagrange

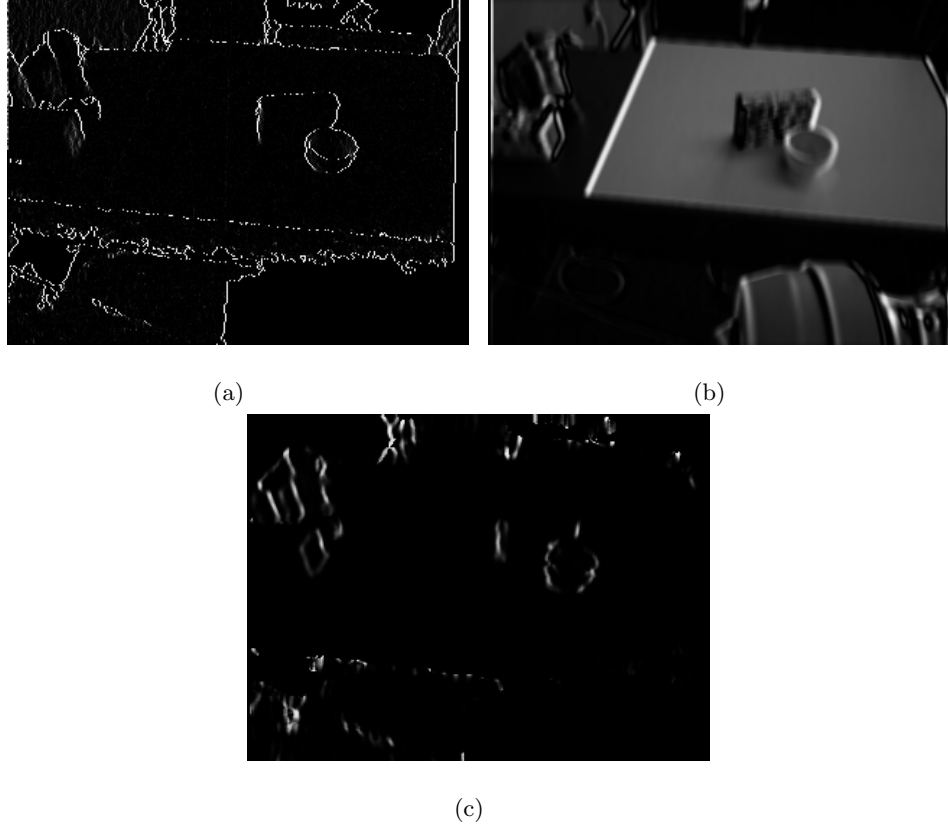


Figure 5.1: The (a) depth, (b) color, and (c) surface normal components of our regularizer for the first frame pair of fig. 5.2. The brighter the pixel, the smaller the component’s contribution to the regularization weight at that location. The higher the regularization weight, the more smoothing occurs between the pixel and a neighbor. In this figure that neighbor is horizontal; we also compute vertical regularization weights.

equations, one for each component of the flow  $\vec{u} = (u, v, w)$ . The Euler-Lagrange equation

for  $u$  is:

$$\begin{aligned}
0 &= \frac{\partial F}{\partial u} - \frac{\partial}{\partial x} \frac{\partial F}{\partial \frac{\partial u}{\partial x}} - \frac{\partial}{\partial y} \frac{\partial F}{\partial \frac{\partial u}{\partial y}} \\
&= \psi'((\Delta_t I)^2) 2(\Delta_t I) \frac{\partial I_2}{\partial x}(\vec{x} + \vec{u}) \\
&\quad + \mu(\vec{x}) \psi'((\Delta_t Z - w(\vec{x}))^2) 2(\Delta_t Z - w(\vec{x})) \frac{\partial Z_2}{\partial x}(\vec{x} + \vec{u}) \\
&\quad - \frac{\partial}{\partial x} (\alpha \psi'(|\nabla \vec{u}|^2) 2 \frac{\partial u}{\partial x}) - \frac{\partial}{\partial y} (\alpha \psi'(|\nabla \vec{u}|^2) 2 \frac{\partial u}{\partial y}) + 2\gamma u \\
&\quad [\text{rewrite } u \rightarrow u + du] \\
&\approx \psi'_D \frac{\partial I_1}{\partial x}(\vec{x}) (\Delta_t I + \frac{\partial I_1}{\partial x}(\vec{x}) du + \frac{\partial I_1}{\partial y}(\vec{x}) dv) \\
&\quad + \mu(\vec{x}) \psi'_Z \frac{\partial Z_1}{\partial x}(\vec{x}) (\Delta_t Z + \frac{\partial Z_1}{\partial x}(\vec{x}) du + \frac{\partial Z_1}{\partial y}(\vec{x}) dv - dw) \\
&\quad - \alpha \psi'_S \nabla^2(u + du) + \gamma(u + du)
\end{aligned}$$

where we define

$$\begin{aligned}
\psi'_D &= \psi'((\Delta_t I)^2), \\
\psi'_Z &= \psi'((\Delta_t Z - w(\vec{x}))^2), \\
\psi'_S &= \psi'(|\nabla \vec{u}|^2).
\end{aligned}$$

Then the SOR update step, with  $\omega$  the SOR parameter, is

$$\begin{aligned}
du &\leftarrow (1 - \omega) du + \omega \frac{num}{denom} \\
\text{with } num &= -\psi'_D \frac{\partial I_1}{\partial x}(\vec{x}) (\Delta_t I + \frac{\partial I_1}{\partial y}(\vec{x}) dv) \\
&\quad - \mu(\vec{x}) \psi'_Z \frac{\partial Z_1}{\partial x}(\vec{x}) (\Delta_t Z + \frac{\partial Z_1}{\partial y}(\vec{x}) dv - dw) \\
&\quad + \alpha \nabla^2(u) + \alpha \nabla_N^2(du) - \gamma u \\
denom &= \psi'_D \frac{\partial I_1}{\partial x}(\vec{x})^2 + \mu(\vec{x}) \psi'_Z \frac{\partial Z_1}{\partial x}(\vec{x})^2 - \alpha \nabla_P^2(du) + \gamma
\end{aligned}$$

where we approximate the image Laplacian of  $du$  (weighted by  $\psi'_S$ ), using Bruhn et al.'s [28] discretization, by  $\sum_{\vec{y} \in N(\vec{x})} \frac{(\psi'_S(\vec{x}) + \psi'_S(\vec{y}))}{2} \lambda(\vec{x}, \vec{y}) (du(\vec{y}) - du(\vec{x}))$  and define  $Lap_N(du) = \sum_{\vec{y} \in N(\vec{x})} \frac{(\psi'_S(\vec{x}) + \psi'_S(\vec{y}))}{2} \lambda(\vec{x}, \vec{y}) du(\vec{y})$  and  $Lap_P(du) = -\sum_{\vec{y} \in N(\vec{x})} \frac{(\psi'_S(\vec{x}) + \psi'_S(\vec{y}))}{2} \lambda(\vec{x}, \vec{y})$ . (The actual code does a further approximation of each  $\psi'_S(\vec{x}) + \psi'_S(\vec{y})$  term by one or the other single-pixel psi value, depending on  $\vec{x}$ . This was part of Liu's code.) The derivation for  $v$  is analogous and omitted.



The Euler-Lagrange equation for  $w$  is

$$\begin{aligned}
0 &= \frac{\partial F}{\partial w} - \frac{\partial}{\partial x} \frac{\partial F}{\partial \frac{\partial w}{\partial x}} - \frac{\partial}{\partial y} \frac{\partial F}{\partial \frac{\partial w}{\partial y}} \\
&\quad [\text{making the approximation that } \frac{\partial \beta}{\partial \vec{x}} \approx 0] \\
&= \mu(\vec{x})\psi'((\Delta_t Z - w(\vec{x}))^2)2(\Delta_t Z - w(\vec{x}))(-1) + 2\gamma\eta(\vec{x})w \\
&\quad - \frac{\partial}{\partial x}(\alpha\psi'(|\nabla \vec{u}|^2)2\beta(\vec{x})\frac{\partial w}{\partial x}) - \frac{\partial}{\partial y}(\alpha\psi'(|\nabla \vec{u}|^2)2\beta(\vec{x})\frac{\partial w}{\partial y}) \\
&\quad [\text{rewrite } w \rightarrow w + dw] \\
&\approx \mu(\vec{x})\psi'_Z(\Delta_t Z + \frac{\partial Z_1}{\partial x}(\vec{x})du + \frac{\partial Z_1}{\partial y}(\vec{x})dv - dw)(-1) \\
&\quad - \alpha\psi'_S \nabla^2(w + dw) + 2\gamma\eta(\vec{x})(w + dw).
\end{aligned}$$

The SOR update step for  $w$  is

$$\begin{aligned}
dw &\leftarrow (1 - \omega)dw + \omega \frac{num}{denom} \\
\text{with } num &= \mu(\vec{x})\psi'_Z(\Delta_t Z + \frac{\partial Z_1}{\partial x}(\vec{x})du + \frac{\partial Z_1}{\partial y}(\vec{x})dv) \\
&\quad + \alpha \nabla^2(w) + \alpha\beta(\vec{x})\nabla_N^2(dw) - \gamma\eta(\vec{x})w \\
denom &= \mu(\vec{x})\psi'_Z - \alpha\beta(\vec{x})\nabla_P^2(dw) + \gamma\eta(\vec{x}).
\end{aligned}$$

### *Practical Issues*

We optimize over a scale-space pyramid. At each scale we use the fixed-point optimization suggested by [24] to iteratively linearize this system of equations as a function of a small change  $d\vec{u}$  in the flow field, and use a parallel sparse linear solver. The smoothing used to downsample the image and upsample the flow field is done in 2-D, making it not ideal for scene flow computation since neighboring pixels are not always close in 3-D. However, if we assume small total motion this is still a reasonable approximation.

The most variable factor in the speed of flow estimation is how long the optimizer takes to converge. Convergence is affected by the number of variables being estimated, by the amount of smoothing (smoothing slows convergence), and by the strength of the flow magnitude penalty (we find that a large penalty slows convergence).

The biggest factor in the numerical stability of optimization is the magnitude of derivatives of the depth map. In general the largest depth-map derivatives are much larger than the largest image derivatives; image edges tend to be blurred over several pixels, whereas

depth discontinuities reported by the Kinect do not. (Measured depth edges are noisy in different ways.) Large derivatives break the smoothness assumptions made when the depth data term is linearized, which happens eventually even in methods that use “nonlinearized” data terms (in an inner loop). At present we fight numerical instability by limiting the magnitude of the depth derivative used in optimization. This slows convergence slightly but avoids having the flow bounce back and forth across the true value each time image gradients are recalculated.

### 5.3.5 *Object Segmentation from Motion with Scene Flow*

One motivation for this work is autonomous segmentation in indoor scenes. Some of this can be accomplished with object instance recognition, but in general some objects will be previously unseen and some will be heavily occluded, necessitating active vision. Once the robot has manipulated part of the scene, for example by pushing a surface, it needs to perform motion segmentation and estimate object boundaries. In sec. 4.5 we discuss a system for rigid object segmentation from motion that works by comparing views of the scene before and after a manipulation. That algorithm is applicable to any two scenes, but is slow due to the need for a very large number of sample points, and its method for selecting a small number of candidate motions from a long list is error-prone.

We find that the candidate selection step is the hardest part of the motion segmentation process, partly due to the large number of similar motions that are proposed. If seed points for motion fitting are densely sampled in space, many tuples of correspondences can be obtained by shifting each point of another tuple slightly in a given direction, and these will result in many similar motions. In order to avoid missing the true motion, in sec. 4.5 we allow each point in one scene to correspond to any of hundreds in the other scene. Since then, to make motion candidate selection faster and more robust, we have experimented with limiting the set of possible point correspondences for each pixel to one. Two ways to produce a single correspondence per point are rigid point cloud alignment—using, e.g., the iterated closest points algorithm (ICP) [15]—and nonrigid motion estimation such as that provided by optical flow or scene flow. A major limitation of ICP is that it must be run separately for each region that moves differently. That is, it theoretically requires that segmentation be known. Therefore in this work we use scene flow to choose a single

candidate point correspondence per pixel. Optical or scene flow, while accurate to less than a pixel for some well-behaved datasets, is known to have problems with untextured regions, where flow is determined mostly by smoothness constraints. However, for small motions, such as between consecutive video frames, scene flow is a fast way to find a short list of reasonable-quality point correspondences.

To perform motion segmentation, we run scene flow on two frames and assign each pixel its scene flow as a proposed motion. We run the algorithm of [78] to iteratively fit a rigid motion using RANSAC and remove that motion’s inliers from the seed set. Efficiency improvements in RANSAC are possible given the knowledge that each point has at most one correspondence, so this step is faster than it is when used in the general case. We follow the fitting procedure with MAP inference in an MRF to select a single motion per point while enforcing spatial smoothness of labels.

### 5.3.6 Experiments

We demonstrate our scene flow method on a variety of real data relevant to domestic robotics. Fig. 5.2 shows results for four scene flow algorithms on videos we collected, and fig. 5.4 shows our results on a frame pair from [66], for comparison to their results. The **Liu+DM** algorithm estimates scene flow by first estimating optical flow from color information only using the implementation of Liu [118] (on which we base our scene flow implementation), followed by sampling the depth map  $Z_2$  at the flowed-to location in the image plane. This is a good baseline and is even slightly more sophisticated than the scene flow method of [179], which only *approximated* the depth map in the depth sampling step. The **RGB-D nonrobust** algorithm is our method modified to use quadratic penalties on the data and smoothness terms. This method tends to oversmooth both the optical and z-flow estimates, demonstrating the reason for using robust penalty functions in flow estimation. We then show results from running RGB-D FLOW with isotropic smoothing and with the anisotropic smoothing of eqn. 5.11. Results of our algorithm with isotropic smoothing should be similar to results of running [183]. The parameters of each method were adjusted manually; in most cases the values are the same. The first row of each column shows the two color images. Each column shows estimated flow in xy and in z separately. The xy-flows are shown in the Middlebury color coding [11]. The z-flows are shown using the x axis of

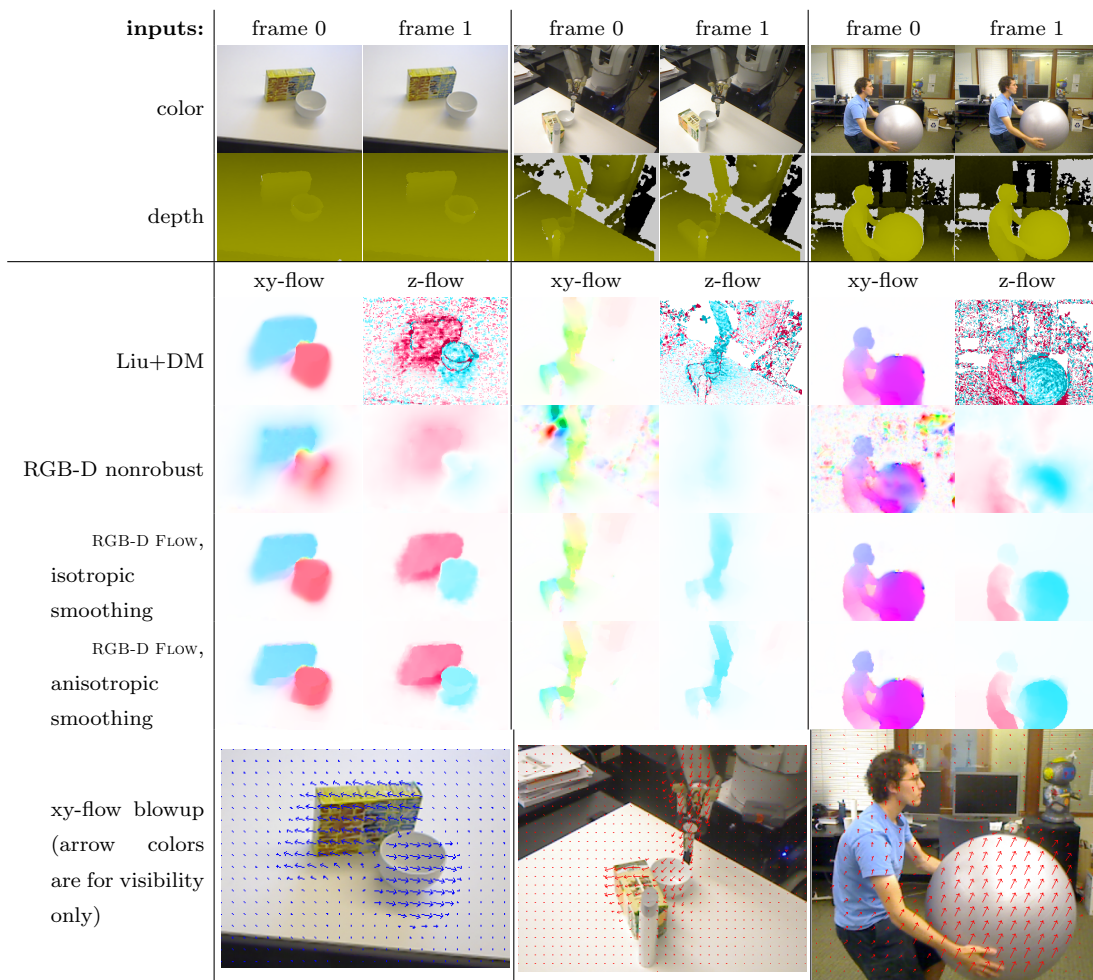


Figure 5.2: scene flow results on three frame pairs: two rigid objects moving; two rigid objects and a robot arm moving; full-body human motion. Gray pixels in depth images show invalid readings. First column for each frame pair is xy-flow; second column is z-flow. Shown in the Middlebury optical flow color coding (for z-flow we use colors along the x axis) with maximum flow (10, 5, and 10) pixels in xy respectively, and all with maximum .01 m in z.

the same coding. Within each column all images are coded using the same maximum flow. For ease of interpretation we include blowups of the xy-flow fields visualized with arrows rather than Middlebury colors.

One of the frame pairs in fig. 5.2 shows human full-body motion. This is intended as a comparison to a method similar to that of Letouzey et al. [116] on data similar to those

they show. Here `RGB-D FLOW` with anisotropic smoothing is similar to Letouzey’s method but uses a more sophisticated regularization (and robust penalty functions).

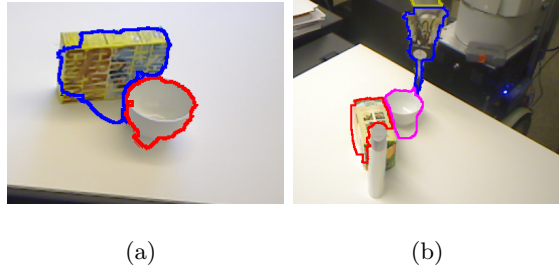


Figure 5.3: rigid object segmentation results on two frame pairs from fig. 5.2. Different motions are shown in different colors.

In fig. 5.3 we show segmentations obtained using the results of `RGB-D FLOW` with anisotropic regularization on the two of these frame pairs that involve rigid motion. In both we obtain the correct number of motions and good segmentations. The shampoo bottle in the second scene doesn’t move and so is identified as background.

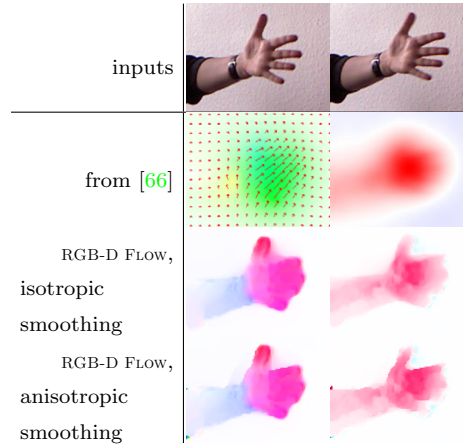


Figure 5.4: scene flow results on a frame pair from [66]. First column: xy-flow; second column: z-flow. Shown in the Middlebury optical flow color coding (for z-flow we use colors along the x axis) with maximum flow 4 pixels in xy, .005 m in z. For comparison we include part of the first row of fig. 3 of [66] (the best-looking of their results). The color scheme, of course, differs from ours.

Fig. 5.4 demonstrates our method on the data used by Gottfried et al. [66], who published

their data. They didn't register the color and depth images in the published data, so we approximately registered them by shifting the depth images in 2-D by a manually chosen number of pixels so that the alignment looks reasonable. Also, these data give depth in unspecified units, so our results may be worsened slightly due to our not adjusting the depth uncertainty model to use a unit other than millimeters.

Our method results in similar or better xy- and z-flow to the optical-flow-based method on all videos shown. It is clear also that using quadratic penalties worsens flow estimation. Anisotropic smoothing appears helpful on most scenes. On the first frame pair of fig. 5.2, the outline of the box is noticeably cleaner with anisotropic than with isotropic smoothing, and on the second frame pair, the outline of the robot arm is also very clean. On the third frame pair, anisotropy makes the outlines of the bottom of the ball and the bottom of the person clearer, although it's not clear whether this improves flow accuracy on the person's left leg. On the frame pair of fig. 5.4, the anisotropic regularizer makes the hand outline clearer particularly around the pointer finger. We judge that anisotropy is usually an improvement on the close- and medium-range scenes in which we are interested.

The runtime of RGB-D FLOW varies between 8 and 30 seconds per frame pair at 320x240 resolution; the variance is due mostly to the varying amounts of smoothing necessary in different scenes, since we optimize to convergence rather than to a fixed number of iterations. The implementation is on the CPU except for the sparse linear solver [136]. We experimented with speeding up the algorithm by instead optimizing with SOR and limiting its iterations, as was done by [183], and did not find this to provide enough smoothing on our data.

Videos showing scene flow for the whole of the second sequence shown in fig. 5.2 and others are available from <https://www.cs.washington.edu/node/8501/>.

### 5.3.7 Summary and Future Work

We have developed a state-of-the-art scene flow estimation algorithm that uses RGB-D data. We have shown that using dense depth information improves estimation of scene flow, the incremental 3-D translation of scene points. Further, scene flow can be used for motion segmentation, even for objects without texture.

The largest problem with the current algorithm is its lack of occlusion handling, which

forces pixels that disappear to take on large estimated movement in order to satisfy the data constraints. This is visible in the results for the human-motion frame pair of fig. 5.2, where pixels around the edges of the ball are assigned large flows. We are working with a formulation that adds occlusion handling to the scene flow optimization in a way that makes use of our RGB-D data. This extended formulation introduces more interactions between variables than are present in the formulation given in this thesis, making each linear system harder to solve than those described here and decreasing optimization speed, and is work in progress.

Representing the uncertainty of estimated flow would speed the sampling step of rigid motion fitting by allowing us to preferentially select seed points with low flow uncertainty. Work such as [133, 27, 103, 61] has suggested multiple ways to measure the uncertainty of optical flow; these are all extensible to scene flow, although it remains to be seen whether any can be made efficient. Perhaps more important is to aggregate the results of two-frame motion segmentation over many pairs of frames, since two frames do not appear to be enough for reliably high-quality segmentation (cf. fig. 5.2). This will aid in using flow in fully online vision algorithms, which have some idea of when their information is incomplete.

We plan to use scene flow for interactive object segmentation and manipulation. When objects are being moved continuously, differential motion estimation between consecutive frames should provide an improvement over the motion estimation of sec. 4.5, which requires two static scenes.

#### **5.4 Segmentation from Human Demonstration**

All of the segmentation methods discussed so far in this chapter have limitations. Because there is no data association step in change detection, two objects that move between two scenes but that are physically close in one scene can't be segmented separately in that scene, even if they're separate in the other. Also, as mentioned before, if we see an area of space that we didn't see in a previous visit, we can't get change detection information for it. The dense feature-matching-based approach we have described requires that objects move mostly out of the space they occupied before moving, as well as that the same surfaces be visible before and after the motion. Segmentation from motion estimation is problematic due to the large amount of smoothing necessary in any dense motion estimation algorithm, even



one relying partly on matching of distinctive sparse features, such as [25]. In this section we describe an attempt at circumventing all these issues caused by autonomy by relying on a human to provide us with a reasonably structured input stream for segmentation. The effort required of the human teacher is much less than marking up a 3-D model with object boundaries would require, but permits our algorithm to make several simplifying assumptions.

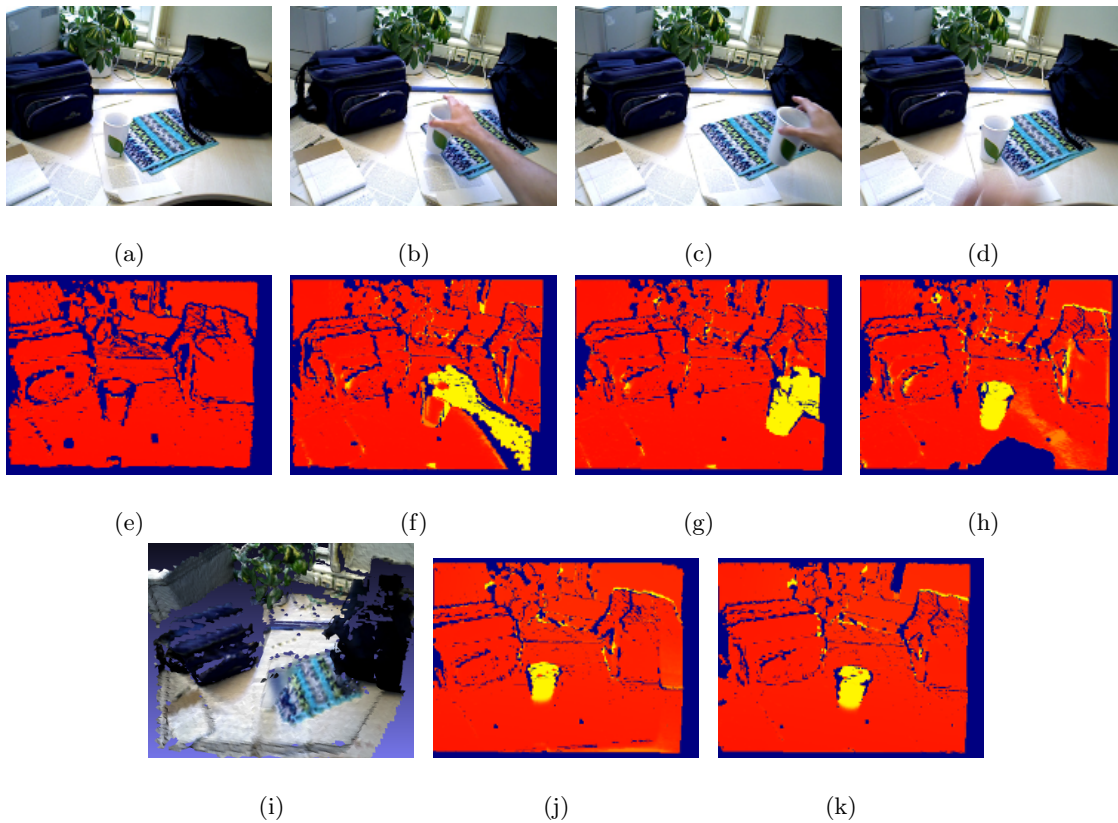


Figure 5.5: change detection and segmentation from a human demonstration of a mostly textureless cup. (a) - (d) frames from the beginning, middle and end of a segmentation demonstration; (e) - (h) the result of differencing these frames against the demonstration background map at each time (red = unchanged, yellow = changed); (i) the background map at the end of the demonstration; (j), (k) the result of differencing the first and last frames of the demonstration against this background map.

We here show how to use a human demonstration of object boundaries (hereafter a



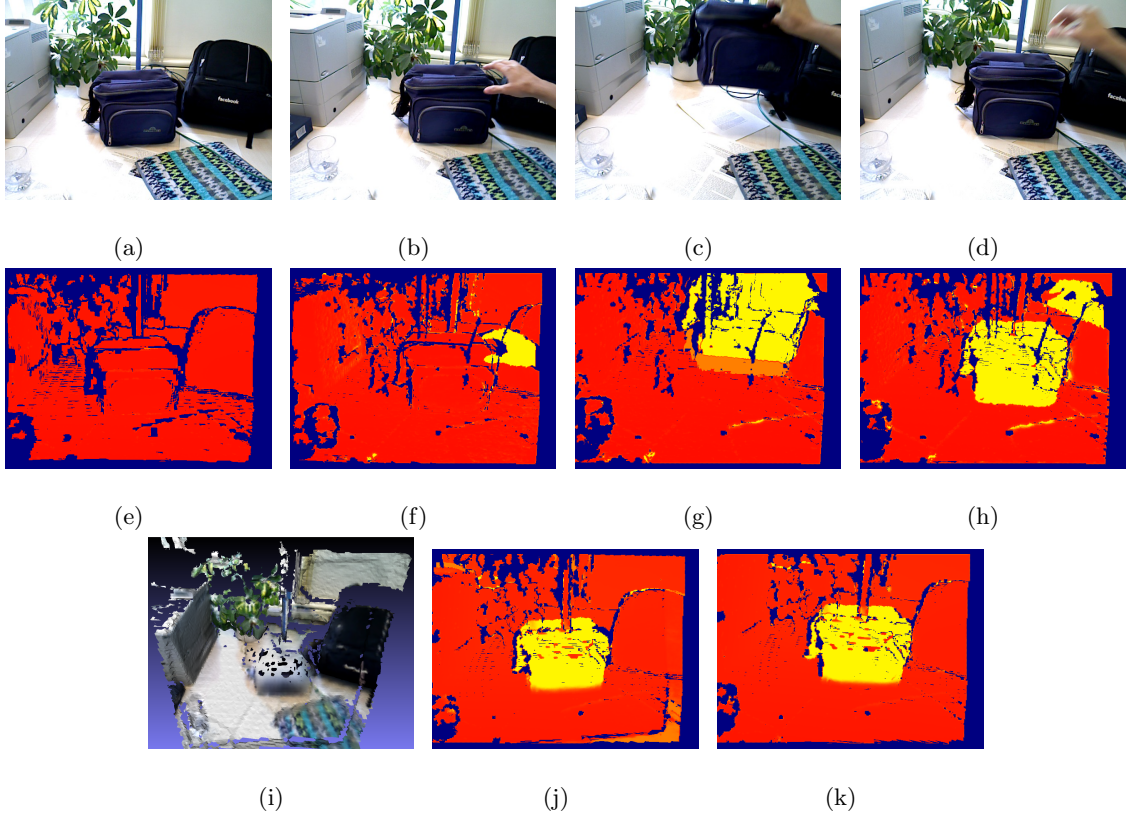


Figure 5.6: change detection and segmentation from a human demonstration of a lunchbox, which is actually slightly nonrigid. (a) - (d) frames from the beginning, middle and end of a segmentation demonstration; (e) - (h) the result of differencing these frames against the demonstration background map at each time (red = unchanged, yellow = changed); (i) the background map at the end of the demonstration; (j), (k) the result of differencing the first and last frames of the demonstration against this background map.

“demonstration”, which comes in the form of a subsequence of a video captured by the robot’s camera) to provide the 3-D extent of a single object in the scene, as well as that of the empty space between the object and the background. The only user interaction required is picking up an object and putting it back down. In order to achieve this level of autonomy, we must place some restrictions on the interaction. The user should move one object out of the space it previously occupied, then replace it close to its original location (with a parameterized maximum distance). There can be camera motion during

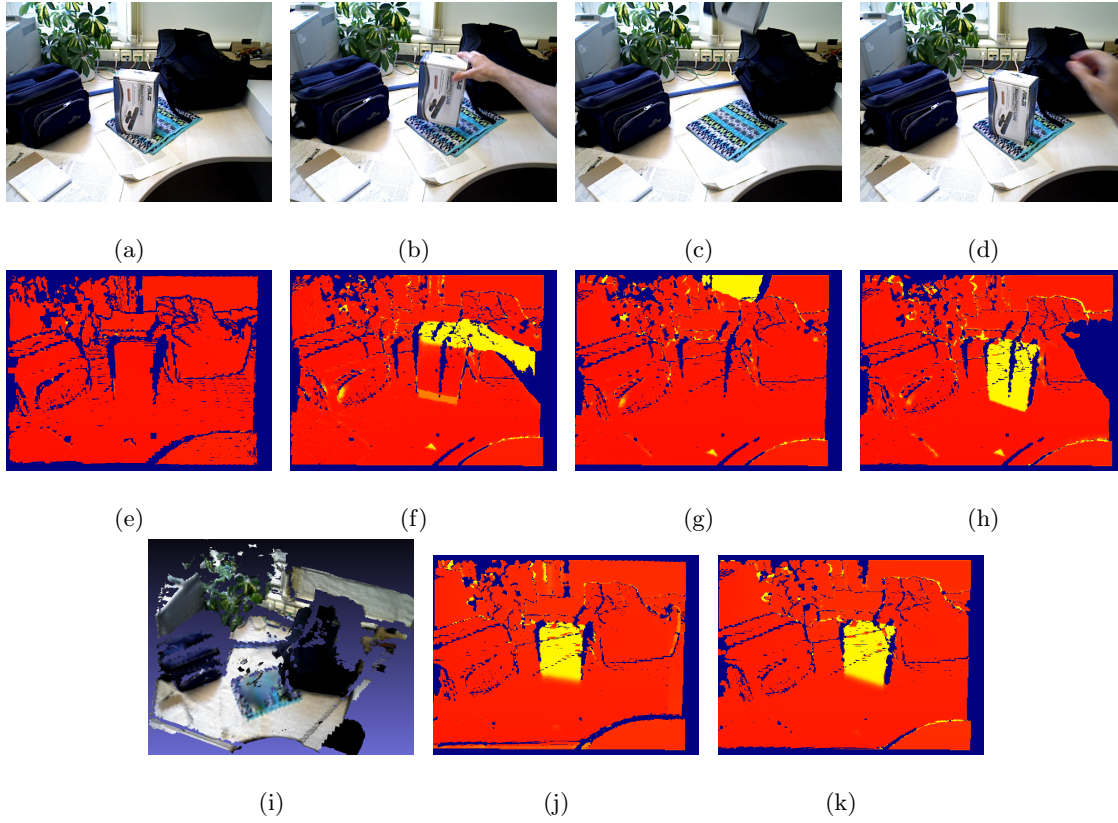


Figure 5.7: change detection and segmentation from a human demonstration of a box. (a) - (d) frames from the beginning, middle and end of a segmentation demonstration; (e) - (h) the result of differencing these frames against the demonstration background map at each time (red = unchanged, yellow = changed); (i) the background map at the end of the demonstration; (j), (k) the result of differencing the first and last frames of the demonstration against this background map.

the demonstration. The first row of fig. 5.5 shows frames from a demonstration video: in fig. 5.5(a) the interaction has not yet happened; in fig. 5.5(b) the demonstrator has picked up the cup; in fig. 5.5(c) the cup is being moved; and in fig. 5.5(d) the demonstrator has put the cup back down and is leaving the frame. (Figs. 5.6 and 5.7 are formatted in the same way as fig. 5.5.)

To ensure that demonstrations always provide unambiguous segmentations, we require that the object be put down in front of all possible occluders from the point of view of the camera during that demonstration (which does not preclude other objects from moving in

front of that object’s location either before or afterward). If the system were running on a robot, the robot could alternatively attempt to grasp and move objects itself to learn their shape. However, this would entail potentially many failures to move the objects properly and would make the associated vision problems harder.

This method applies during recording of a video that is concurrently being used to perform online SLAM to create a static 3-D reconstruction of the scene. We determine the start of a demonstration to be a video frame in which we see a large spatially connected segment of pixels that have moved with respect to a recent frame (e.g. three frames prior) and whose motion cannot be explained by the background motion computed by map alignment, and we designate an end to the demonstration when the number of changed pixels with respect to the pre-demonstration map ceases to change. At each demonstration frame, we identify changed points with respect to the current background map (shown in yellow in figs. 5.5(e) through 5.5(h) and in the second rows of figs. 5.6 and 5.7), and add all *other* points to the background map. This means that we add both pixels that agree very well with the map and those that were previously occluded or unviewed. If we add only previously unviewed pixels (which overlap the 3-D map very little), the resulting map contains many sharp edges at a small scale. We also find that we need to decrease the confidence of each TSDF voxel at each frame before adding to the map, or the map changes extremely slowly. At the end of a demonstration, if the object has been outside the space it originally occupied for long enough, it no longer exists in the background map. See figs. 5.5(i), 5.6(i) and 5.7(i) for the background maps resulting from these three demonstrations.

Once we have determined that the demonstration has ended, we segment out the object in the first and last frames of the demonstration. Assuming that at some point the object moves entirely out of the space it previously occupied and that no part of the user is visible at the start or end of the demonstration, this can be accomplished simply by detecting change in these two frames with respect to the background map, which no longer contains the object. (Because of the probabilistic nature of the volumetric mapping technique we use, there may be pieces of the object left in the background map. Our technique for eradicating these from the volumetric representation requires the integration of a large number of frames not containing the object, and so depends on the length of the demonstration video. These pieces might be relatively easily removed from a mesh created from the volumetric map, by thresholding on connected-component size. However, here we mainly work with voxels

rather than meshes.) We reoptimize the relative pose between the first frame and the background map at this point to correct for drift during mapping, which occurs both due to the lack of loop closure in our volumetric mapping and because we are abusing a static SLAM system to build a map of a changing environment, in that the modified frames we add to the map during a demonstration do not all contain the same surfaces. Figs. 5.5(j) and 5.5(k) show the segmentations of the object demonstrated in that figure in the first and last frames of that demonstration; similarly see object segmentations in figs. 5.6(j) - 5.6(k) and 5.7(j) and 5.7(k).

A possibly simpler method of building a background map during a demonstration would be to identify a frame in the middle in which the object is entirely out of the image (thus requiring that there be such a time) and detect change in the first and last demonstration frames with respect to this frame. The identification might be done by finding a point at which there is very little change with respect to a few frames prior, or by identifying a frame in which no surface is in front of its location in the first demonstration frame. Despite being simpler than our method, however, neither of these is as robust: the first algorithm breaks if the user holds the object steady before replacing it, and the second breaks if the user removes the object from opposite the camera.

Finally, using these segmentations we estimate the rigid motion of the object during the demonstration, with the goal of using the motion to update and create object models (although this is not yet implemented). We accomplish this with a RANSAC algorithm, repeatedly sampling triples of point correspondences between frames just before and just after demonstration, when we know the human is not present in the scene. We have experimented with both the exhaustive approach of [78] and the scene-flow-based approach of [79] for generating correspondences, and find that we require many fewer iterations of sampling when using only one correspondence per point, as given by scene flow. We see on the order of ten sampling iterations when using flow and on the order of ten thousand when using exhaustive correspondence selection. Therefore we use flow in the experiments presented here. Both algorithms require a limit on how far the object can move during the demonstration, the exhaustive approach in order to limit the number of possible correspondences (which affects speed but not correctness) and the flow approach because of the limits of image-based spatial regularization (which affects correctness).

At the end of the demonstration we create a volumetric map from the segment of the

object in the final demonstration frame. It would be possible to also add to this model the object as seen in the first demonstration frame, since we have the rigid transformation between the two. It would not be possible to incorporate intermediate frames into the model, due to the difficulty of segmenting the object apart from the human. If this demonstration component were integrated fully with our mapping system, object models resulting from demonstrations would be used in the same way as models acquired otherwise.

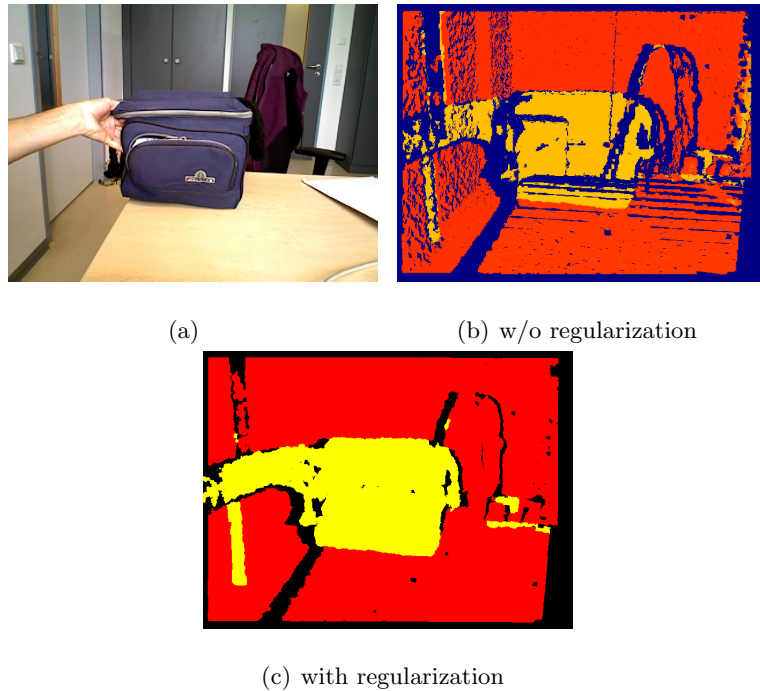


Figure 5.8: the use of nonlocal change detection information in automating detection of begin and end points of episodes such as segmentation demonstrations. We show the labeling of points in an incoming frame (a) as changed and unchanged with respect to a background map (not shown), both before (b) and after (c) spatial regularization using information not available to pixelwise change detection. Results with regularization are significantly more consistent over time due to the many pixels in the rendered background map that have no valid depth values (shown as dark blue in (b)). This frame is in the middle of a demonstration of a lunchbox.

We use counts of likely-changed pixels to determine at what frames a demonstration starts and stops. These counts are affected by what part of the scene background the object is in front of at different points in time, as large parts of the background map do not have

valid depth measurements and so cannot provide change detection information. This is illustrated in fig. 5.8. We therefore define a Markov random field with 4-connections among all pixels that have valid depth in the frame, regardless of whether they have valid depth in the map, with labels *changed*, *unchanged* and *unknown* at each pixel, and run inference in it to get a final count at each frame. We can thereby extend the changed/unchanged labeling provided by pointwise change detection to pixels without change detection information. This improves the consistency of our counts over time and significantly improves our ability to detect changes in the amount of surface that is moving.

### 5.5 Summary

In this and the preceding chapter we have developed a variety of tools for segmenting objects using different motion cues. We have described three algorithms for robotic segmentation using motion cues. Our change-detection-based algorithm assumes large motion of objects has occurred while the robot is away from the scene. A method based on dense feature matching allows us to use the robot to cause motion without making strong assumptions on the form of that motion, since such assumptions would often be violated due to the difficulty of causing a desired motion autonomously. Robot interactivity can only be achieved if the computer vision that is relied on can be performed online. Therefore, a third algorithm, designed for online processing of video of object motion, uses dense motion estimation and therefore can produce correspondences between surfaces at different times during the motion. These cues are useful in different situations, and none of these segmentation algorithms dominates any other. The first is the most efficient one when objects are widely separated spatially; the second is the most general but doesn't degrade well as parameters are adjusted to speed it up, since that means ignoring many potentially good motion candidates; and the third requires that motion be small and has more problems than the others do in dealing with scenes with large depth discontinuities, due to the 2-D nature of the (image-based) regularization. In order to work around these weaknesses as necessary, we have also described a method for using a semi-structured human interaction with the scene to provide high-quality segmentation information to an observing camera. The interaction required is limited to picking up and putting down an object, with some restrictions on the motion.



## Chapter 6

### ONLINE OBJECT SEGMENTATION AND MODELING

Up to this point we have mostly presented segmentation algorithms that run on 3-D maps, necessitating that they be run only after a video of a scene is taken. The algorithm of sec. 5.3, which estimates motion during a video, does not require maps but is not able to process an entire video; an interesting interval must be selected for it to run on. Now we consider segmenting objects in an online fashion as a video is taken. Our main reason for doing this is that the response of a system has to be real-time to permit close examination of objects, manipulation, or other interactivity, and interactivity with novel objects is one of our goals.

Online 3-D reconstruction techniques have recently become fast and accurate enough to permit building other algorithms on top of them. In this chapter we build on a recent online SLAM method to segment objects during, rather than after, scene reconstruction. We take object outline information from change detection to build 3-D models of rigid objects and represent the scene as static and dynamic components that are updated online during video processing. We can integrate segmentation information from sources other than change detection.

We also take steps toward being able to process continuous video in this fashion. Video from a continuously operating robot will not be nicely segmented into separate “visits” to a scene, so we would like to be able to process whatever video a camera on a mobile robot captures, continuously and for long periods of time. In this chapter we demonstrate the ability to perform online object segmentation and modeling even when objects move (off camera) during the video. As an alternative to change detection between static scenes, we also show how to capture segmentation information for an object from a relatively unconstrained and automatically detectable human interaction. Both these steps increase the generality of the scenarios in which we can perform fully autonomous object segmentation.

## 6.1 Introduction

Over the last few years robotic vision systems have gotten much better at detailed and fast online 3-D reconstruction of static scenes using volumetric approaches running on the GPU [134, 73, 188]. The ability to detect scene changes between detailed 3-D maps was demonstrated in [76]. In this chapter we demonstrate high-quality object segmentation and modeling *simultaneous with* online scene reconstruction.

Imagine that a robot makes recordings of the same scene at several points in time, separated by perhaps a day—long enough for some objects to have moved. We should be able to partition the 3-D reconstruction of the scene into a “background map” of everything that never moves and a set of dynamic rigid objects that move at least once between mapping visits. We use change detection to provide segmentation information, because over long periods of time, objects that can move probably will be moved.

One way in which object segmentation with change detection can fail is in being unable to separate objects that move to or from a spatially adjacent configuration. Another issue is that a robot seeing a previously unobserved region of space has no change detection information for the region. To address these problems, the online mapping framework we discuss in this chapter is able to take in information about segmentation in the current scene that does not come from change detection. We demonstrate this capability by using information from a human-provided movement of a single object that can improve our segmentation of that object and others that it touches. This is complementary to work (e.g. [57]; most recently ours [79]) in which a robot pushes objects to make them move and attempts to segment all moved objects using dense motion estimation. We use some of the same techniques used therein, but as that work demonstrates, segmentation from motion data without very strong regularizers (e.g., a very strong bias for the idea that objects are entirely convex) doesn’t currently work very well. Therefore, here we instead acquire known-correct segmentation information from a human teacher.

Our method avoids requiring visual point features, except during background alignment (see sec. 6.4). One of our goals is to as much as possible build a principled framework around one main sub-model, change detection, used in a variety of ways. All of our modeling happens online: we update our object models at each video frame.

This chapter is organized as follows. In sec. 6.2 we discuss related work. Sec. 6.4 explains



how we use reconstruction and change detection in an online fashion. Sec. 6.5 introduces our object modeling. We discuss using additional segmentation information in sec. 6.6. Finally we mention efficiency in sec. 6.7 and future work in sec. 6.8.

## 6.2 *Related Work*

We cannot list all of the immense amount of work in object segmentation and modeling with vision; we hope to provide a useful sample.

Representing scenes using objects and background is an old idea. In mobile robotics it goes back at least to Biswas et al. [18], who process a set of previously aligned 2-D occupancy grid maps with change detection in an off-line fashion to identify movable objects. Haehnel et al. [71] separate moving and static regions of 2-D laser scans in an online fashion in order to navigate through moving obstacles. They also model moving objects, by extending the 2-D profiles of dynamic regions through a 3-D laser scan. Finman et al. [55] use a combination of 3-D change detection (using depth but not color) and running per-object learned segmentation models on each of a set of maps to extract a set of potential objects that could be input to an object discovery algorithm. Their algorithm is also off-line, but runs on multiple large-scale maps built separately, as does our prior work [76]. Probably the most similar work to this chapter is that of Alimi et al. [4], who create an RGB-D background map after running change detection on many pairs of temporally adjacent views of a scene. Their camera is static and their views are single frames rather than videos. There is also work on linking maps of objects and background using semantic relations: in [194], Wurm et al. analyze a laser scan to identify objects and background, using the table plane assumption and the assumption that objects are all spatially separated to avoid solving segmentation. They create octree maps of objects and background separately, potentially at different spatial resolutions, and connect them with semantic concepts such as “supported-by”. Salas-Moreno et al. [154] match the scene to previously built geometric object models at each frame during reconstruction in order to build a separate information layer with object poses and labels. They update the poses of matched objects online as the 3-D reconstruction changes.

Online motion segmentation is also an active area of research. Pundlik and Birchfield [142] match point features in RGB-D video as input to sparse motion segmentation, and

use spatial regularization over pixels to extend the segmentation to a dense one. Relatedly, Roussos et al. [150] perform motion estimation and reconstruction of 3-D rigid objects from RGB video in an off-line way, using recent optimization techniques to allow them to include all pixels in the optimization objective.

This chapter improves on previous work that segments objects using change detection given videos of multiple visits to a scene, assuming the scene is static during each visit. We can use static SLAM to build a static map from each video. Change detection run over pairs of these maps gives us some information about the geometry of movable objects in the scene. We can merge the unchanged parts of each map into an overall “background map”; the other segments from all scenes are models of moved objects. This approach was taken in [77, 55]. In this work, we instead perform change detection in an online fashion, comparing each new video frame to a previous map *while running SLAM* instead of mapping each scene before detecting changes. There are three major advantages to the online approach. Firstly, Newcombe et al. [134] demonstrate improved alignment performance during SLAM from aligning new frames to existing maps rather than aligning frames to each other in traditional visual odometry style. In experiments not detailed herein we have found a similar effect when aligning frames from one video to a map made from another video for change detection purposes. Secondly, if we perform change detection during reconstruction rather than afterward, we can improve the quality of models of objects present in the current scene (the scene that we’re processing a frame at a time). Because we know, given partial object views, the approximate sizes of objects, we can make high-resolution models of small regions including these objects, while mapping the rest of the scene in relatively low resolution to save memory and time. Thirdly, we want to extend this work to involve *active segmentation*, in which the robot can move the camera and/or objects being modeled in order to get more informative views. This is only possible if objects are modeled in real time.

The research contributions of this chapter are as follows. We combine the idea of making separate maps for static background and dynamic objects with recent advances in online reconstruction. We apply frame-to-map alignment, shown by Newcombe et al. [134] to improve single-map alignment, to alignment between multiple maps. We discuss what information change detection gives us for the purpose of modeling our knowledge about object boundaries.

### 6.3 Aligning Frames to a Map

Our main building block in sec. 4.3, where we use change detection for segmentation, is a change detection algorithm operating on one 3-D map and one RGB-D frame. One weakness of that work is that each map is reconstructed fully in order to align the frames of each map to each other map. The global consistency of static SLAM is poor enough that these frame-to-map alignments can be improved. In this work we do so by aligning each frame of a video separately to a map made from previous videos before running change detection. We find empirically that this modification improves change detection and dense alignment.

In ch. 4 we collect a video of each of two visits to a scene, build a map of each video with a SLAM technique (which did not have to be online SLAM), then align the two maps using visual point feature matching (as a proxy for better techniques that are available to systems that actually run constantly). In this chapter, we collect a video of a visit, build a map from it with any SLAM technique, then, during recording of the second video, align each incoming frame separately to the existing map, initially using point feature matching (again only because of its simplicity for research purposes) and subsequently using dense techniques.

In ch. 8 we use change detection to score hypothesized alignments between partial object models. We can go further and directly optimize a change-detection-based objective for alignment of partial models. (The term *alignment* refers to finding a rigid transformation between two partial models such that the same points on each are brought to the same 3-D pose.) Direct optimization of a change-detection-based objective can replace the ICP-based alignment technique used in ch. 8; it is slower by an order of magnitude or more, but in our experiments (not discussed here) works at least as well.

Given two partial models of a rigid object, we can use scene differencing to create a gradient-based alignment algorithm by defining a scalar-valued objective function in terms of differencing results. A similar technique was used in [111] with a depth-only sensor model. The algorithm is comparable to ICP: ICP can also be thought of as optimizing a (simplistic) sensor model, and its objective function is also highly nonconvex. Our gradient-based algorithm has the advantage of not requiring explicit point correspondences, as ICP does, to evaluate the objective. The score function we minimize, using scene differencing at each surface sample  $s$  in scene  $i$  with respect to scene  $j$  under transform  $X$ , is  $E_{i,j}(X) + E_{j,i}(X)$ ,

where

$$E_{i,j}(X) = \sum_{s \in i} (\max(0.5, \log(p(\mathbf{m}_s = 1 \mid z_s))) + \min(1, \log(p(\mathbf{m}_s = 1 \mid z_s)) + \log(p(\mathbf{m}_s = 0 \mid z_s))))^2. \quad (6.1)$$

This formulation penalizes both large difference in local appearance (the first term in the sum) and high uncertainty (the second term). 0.5 is the value of probability of change that we use for occluded areas; the first term avoids penalizing regions that are occluded under the hypothesized transformation for being invisible. The uncertainty penalty is useful in helping to reject transformations under which the two models overlap not at all. The problem of determining that they do overlap and should be able to be aligned is a separate issue in this formulation. We can remove the uncertainty term when we also wish to decide whether two models represent the same object.

In addition to aligning models of single objects, direct optimization of a change-detection-based objective can be used to align frames from a video to an existing map of a similar scene, as we wish to do in this chapter. One simple way to perform this alignment is to match visual point features found in the new frame to those found in the existing map and compute a transformation between the two 3-D point sets using, for example, Horn’s method [84]. This algorithm can lead to jitteriness or failure in alignment results, however, when features are not distinctive and many false matches occur, when the existing map is inconsistent (alignments among the frames used to build the map are inaccurate), or when there are very few distinctive features in the scene and match counts are low. We have experimented, therefore, with using direct optimization of a differencing-based score for alignment of frames to a map. This local optimization requires an initialization. At each frame, we run a visual-feature-based alignment first, then use both the result of that technique and the estimated alignment of the previous frame as initial estimates for the differencing optimization. We select the optimization result with the better score as the final estimate for the new frame. This significantly reduces both jitter over time and alignment failures during the course of a video. Fig. 6.1 shows intermediate alignment results for each of these algorithms on some of the datasets used in ch. 8. In particular, the table and the dinosaur are much better aligned with this method than with either feature-matching-based method shown in the figure. The map used here was built with RGB-D Mapping (which was introduced in ch. 2.2) and consists of surfels. These results are taken 40 frames into the

new video, so that errors have had a chance to build up over time. Full videos of the results snapshotted in the figure are available at <http://mobilerobotics.cs.washington.edu/projects/online-segmentation/onlineDiffingAlignmentMovies.html>.

The alignment optimization used in the Patch Volumes system (see ch. 2.2) is similar to this differencing optimization at a high level, in that both objective functions are formulated as sums of squares over pixels, combining depth and color information. The objective function used in fig. 6.1 is a modification of that of eq. 6.1 to account for the large number of pixels involved. To compare image  $i$  to scene  $j$ , we minimize  $F_{i,j}(X) + F_{j,i}(X)$ , where

$$F_{i,j}(X) = \sum_{R \in \mathcal{R}} \left( \sum_{p \in R} A_{i,j}(p) \right)^2, \quad (6.2)$$

where  $\mathcal{R}$  is a set of image regions that partition the space of pixels in image  $i$  and  $A_{i,j}(p)$  stands for the quantity inside the square in eq. 6.1, indexed by pixel  $p$  in  $i$ . This partitioning is done to reduce the number of terms in the sum of squares in order to speed up the least squares solver on the CPU, since this code has not been GPUized yet. There is a speed/accuracy tradeoff in selecting the number of regions. Currently we use 48 regions. We optimize using Levenberg-Marquardt with finite-differencing gradients, since there isn't an analytical gradient for our change detection model.

The Patch Volumes alignment routine runs on the GPU, where least-squares optimization with many squared terms in the sum is feasible to do efficiently. The objective being minimized is

$$E(X) = \sum_{pixelsp} \left( \frac{\sigma_z(.4)}{\sigma_z(z)} \epsilon_{P2P}(p) + \Delta I(p) \right)^2, \quad (6.3)$$

where  $\sigma_z(z)$  gives the depth sensor uncertainty as a function of depth  $z$ ,  $\epsilon_{P2P}$  is the point-to-plane error function of [33], and  $\Delta I(p)$  is the difference in intensities between the renderings of the two scenes at pixel  $p$  under  $X$ . This objective is designed to have an easy-to-calculate gradient, which makes it possible for Patch Volumes to use an iterative Gauss-Newton optimization to solve the least squares problem, using the image gradient directly in the gradient calculation rather than relying on finite differencing. The change-detection-based approach currently parameterizes the rotation part of  $X$  using Euler angles, whereas Patch Volumes uses a three-parameter quaternion representation.

In fig. 6.2 we show similar intermediate results for change detection between scenes, using alignment with Patch Volumes for comparison to the methods shown in fig. 6.1. Videos of the GPU-based approach at work are also available at the same URL. This approach results in less jittery alignment than either of the other approaches.

Since the change-detection-based approach is the most mathematically righteous of those presented here, and uses strictly more information than the others, it is the preferred method if we can figure out why it doesn't work as well as Patch Volumes to date. In this vein it would be interesting to determine how much of the difference is due to the regionization of the image in the CPU implementation, how much is due to using finite differencing for gradients, and whether there are any other causes of decreased performance. In the remainder of this chapter we use the Patch Volumes alignment method for speed reasons: our GPU implementation of this method is two orders of magnitude faster than our CPU-based implementation of the change-detection-based optimization.

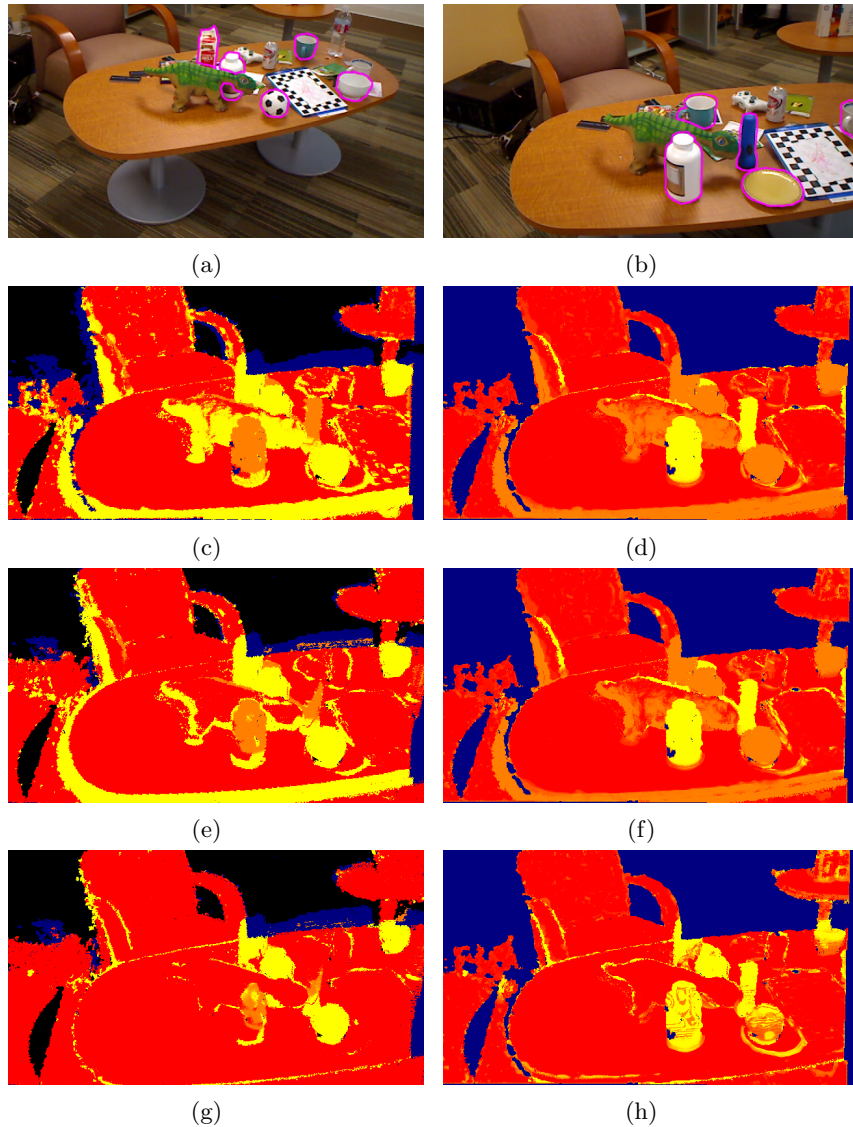


Figure 6.1: comparison of frame-to-map alignment algorithms. We align the frame shown in (b) to a map of the scene shown in (a) for purposes of online change detection. First column: change detection results for the map wrt the frame; second column: change detection results for the frame wrt the map. (c), (d) results for alignment using matching of Calonder et al.’s descriptors [29] computed at FAST [148, 149] keypoints. (e), (f) results for alignment using matching of SIFT descriptors computed at SIFT keypoints. (g), (h) results of an algorithm using both SIFT descriptor matching and local optimization of a change-detection-based score. Black pixels in the first column denote frame pixels to which no map surface projects. Dark blue in both columns denotes pixels with no change detection evidence (no projected surface, or no valid depth reading in the frame, or surface orientation too oblique). In the top row, regions that should show up as changed (yellow) are highlighted in magenta. Almost everything else should be red, signifying that the two scenes at that point are closely aligned.

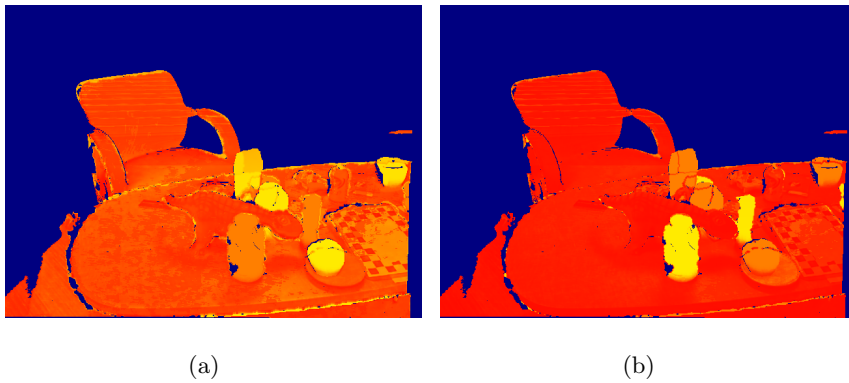


Figure 6.2: Change detection results for the scenes of fig. 6.1 (to which these results should be compared) using the Patch Volumes alignment method. Both results here are shown projected onto the frame rather than the previous scene.



#### 6.4 Online Change Detection

Since change detection can operate on a map and a single frame, we can run it online between a previously built map and each frame of a new scene. This can be used to aggregate change detection results on a map of the new scene that is built after the video is taken, as is done in sec. 4.3, but it can also be used to separate, as the video progresses, our map of the scene into components for the static background and moved objects. Upon receiving each video frame, we update a map of the background as well as models of all movable objects, both those that appear in the new video but not previously processed videos and those that appear in previous videos but not the new one.

The background is represented as a volumetric map, as is each object. We use the Patch Volumes SLAM system, discussed in sec. 2.2, for aligning and adding RGB-D frames to volumetric maps. For efficiency, in this chapter we use statically sized volumes, although we do resize volumes periodically for object modeling.

Our change detection procedure takes as input a relative pose between two scenes (here, an aggregated map  $\mathcal{M}$  and an RGB-D frame  $\mathcal{F}$ ), renders  $\mathcal{M}$  into  $\mathcal{F}$ ’s camera, and compares sensor measurements at each pixel to decide whether the surface appearing at that pixel in  $\mathcal{M}$  also appears at that pixel in  $\mathcal{F}$ . The decision is made mostly on a per-pixel basis, using the “expected” and “observed” attributes rendered from  $\mathcal{M}$  and  $\mathcal{F}$  respectively. For this work we find it is helpful to use the probabilistic noise model for RGB-D sensors that incorporates the uncertainty of the observed depth as well as that of the expected (sec. 3.4). Including the observation uncertainty reduces the model’s certainty (i.e. moves the output probability toward .5) in cases where the observed depth value is not very certain. This is useful both for points with large observed depth (which we mostly avoided using in [76]) and for points near depth boundaries, which can be misaligned in the image plane.

Fig. 6.3 includes an example of change detection. Figs. 6.3(a) and 6.3(d) show frames from two scenes, and figs. 6.3(c) and 6.3(b) show change detection results of each scene with respect to the other.

#### 6.5 Online Mapping and Segmentation

The techniques of sec. 6.4 allow us to create, and continuously update, 3-D models of objects that have moved since a previous visit to the scene. At a high level, this is done

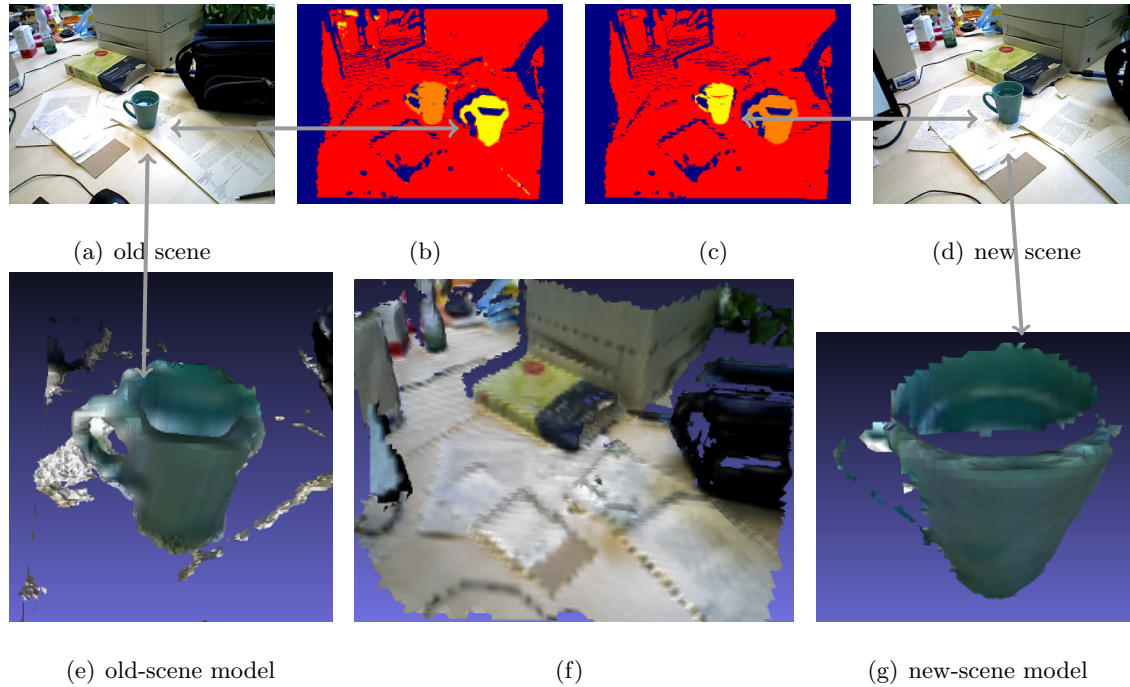


Figure 6.3: online change detection and object modeling. (a), (d) two scenes of a desk on which a cup has moved; (f) the background map and (e), (g) object models resulting from running online change detection on them. (b), (c): change detection results for the first frame of the second video with respect to the map of the first scene and vice versa. Yellow regions are present in one scene but not the other; orange regions are occluded or uncertain; red regions are judged unchanged. Dark blue denotes no information at that pixel in at least one of the two change detection inputs.

by maintaining a separate map for each spatially connected region of surface that change detection marks as changed, and adding nearby pixels to these maps as the video progresses according to a measure of how likely each pixel is to be part of the object modeled by each map. (This determination is also based on change detection.) We create new object maps for pixels that don't match well to any existing object.

We can obtain the initial background for processing a video either from an empty map (in which all voxels are considered “unseen”) or from the result of processing one or more previous scenes. If the background map is not empty, we must initially align the video to the background map. This is done with point feature matching as in [76] but with the second

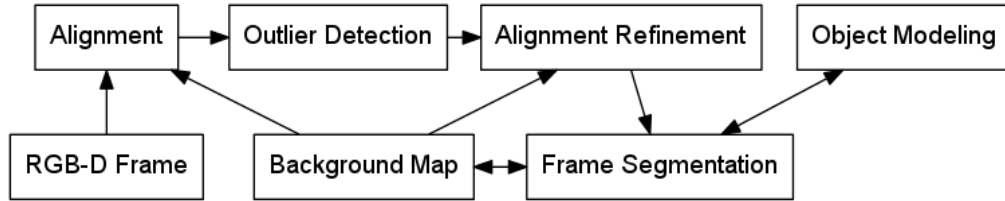


Figure 6.4: our algorithm for processing each frame of a new scene. As each frame comes in from the camera, we align it to the background map, segment it into background and object components using change detection information, and add it to the background map and object models.

map replaced by a single frame, and we ignore incoming frames until we have a good match. From then on, we process each frame with the procedure shown in fig. 6.4. We difference the frame against the background map to perform outlier rejection by identifying pixels likely to be part of movable objects. We use the pose of the previous frame’s camera as the estimated pose for differencing. We mask out outlier pixels, realign with Patch Volumes to get a more accurate alignment, and add the masked frame into the background map. Except for runtime considerations we could run these outlier rejection and realignment steps alternately to some sort of convergence; for now we find a single outlier rejection step sufficient. Next we run change detection again with the updated alignment to update the set of pixels that need to be explained with objects (i.e. outlier pixels with respect to the background scene motion). These pixels are segmented using information from both the current frame and object models we have already made, and each segment is added to one or more existing object models or used to create a new model. See fig. 6.5 for a visualization of this process. We run change detection in both directions to model both newly seen and no-longer-seen objects. After processing an entire scene in this fashion, we have a background map containing none of the objects seen to move as well as a model of each moved object seen in the current scene or in the background model used to process this scene.

When using Patch Volumes alignment, we align the frame to a composite background map created by overlaying the background map and all visible objects from the current scene (done by weighted addition of the TSDFs of multiple maps, with linear interpolation when voxel grids have differing orientations, which they almost always do). In cases when

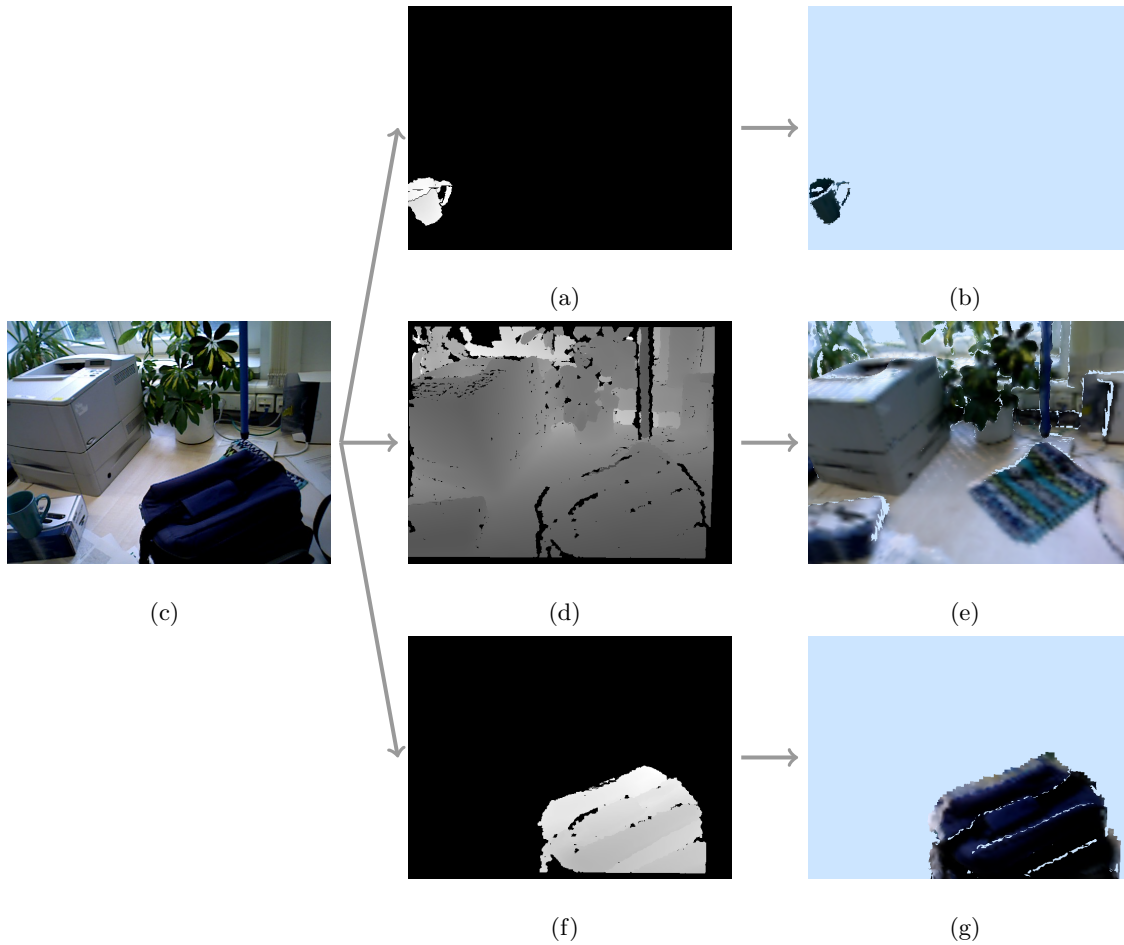


Figure 6.5: our procedure for updating models from a new video frame. Arrows denote information flow. The new frame (c) comes in and is partitioned into segments (a), (d), (f) to be added (after slight processing; e.g., the background segment is extended to cover pixels that are included in non-background segments) to existing object models (b), (g) and the background map (e), all three of which are shown here as renderings from volumetric maps. If either non-background segment here had not corresponded well to an existing object model, a new model would have been created.

the new objects take up a large part of the frame, this measure prevents us from having too few pixels of the background map visible and usable for alignment.

Although Patch Volumes downweights likely outliers during alignment, we find that our outlier rejection works much better in practice. The assignment of pixels to models is one

---

**Algorithm 6.3** the connected components algorithm used to do per-frame segmentation in online modeling.  $\delta z_0$  is the base distance threshold; we use 8 mm.

---

```

 $\mathcal{P} \leftarrow$  the set of pixels with high probability of change
initialize a component for each pixel in  $\mathcal{P}$ 
for each pixel  $p \in \mathcal{P}$  do
  for each pixel  $q \in \mathcal{P}$  within Manhattan distance  $w$  of  $p$  do
     $d \leftarrow$  the Manhattan distance from  $p$  to  $q$ 
    if  $|q_z - p_z| < \delta z_0 \times d$  then
      connect the components containing  $p$  and  $q$ 

```

---

of the most computationally heavy parts of the system. Each existing model is rendered in the frame. Pixels to be added are segmented using 2.5-D connected components (alg. 6.3), and each segment is run through change detection with respect to each model. (This segmentation of the frame reduces the noise concomitant with choosing a model for each pixel separately.) A segment is added to each model that it matches well according to a change-detection-based score: for image segment  $s$  and existing model  $\mathcal{M}$ , we add  $s$  to  $\mathcal{M}$  if  $J_{s,\mathcal{M}} > \tau$ , where

$$J_{s,\mathcal{M}} = \begin{cases} \frac{\sum_{p \in s} e_p (\frac{1}{2} - p_p(m))}{\sum_{p \in s} e_p}, & \sum_{p \in s} e_p > 0 \\ 0, & \sum_{p \in s} e_p = 0 \end{cases}.$$

Here  $e_p$  is Boolean and is 1 iff the sum of the magnitudes of all log-probabilities used in change detection is nonzero at pixel  $p$  (i.e. if there is any evidence at the pixel), and  $p_p(m)$  is the probability of change at pixel  $p$ . The value of  $J$  is in  $[-.5, .5]$ ; we apply a threshold  $\tau \in [.3, .4]$  that depends on the parameters of the change detection model and the amount of noise in the maps (which is a function of map resolution and alignment error). If a segment is added to multiple models, those models are merged (and resized if necessary). The merge operation relies on the form of the volumetric maps: each voxel stores a signed distance and a weight that is essentially the number of times the voxel has been viewed. Merging two volumetric maps thus consists of adding the weights at each voxel (with one map represented by interpolated samples, since maps' axes aren't in general aligned) and performing a weighted sum of the two signed distances and colors at each voxel.

Fig. 6.3 gives an example of the results of processing two scenes in this fashion. We start with an empty map, run the scene of fig. 6.3(a) through the above procedure to get a

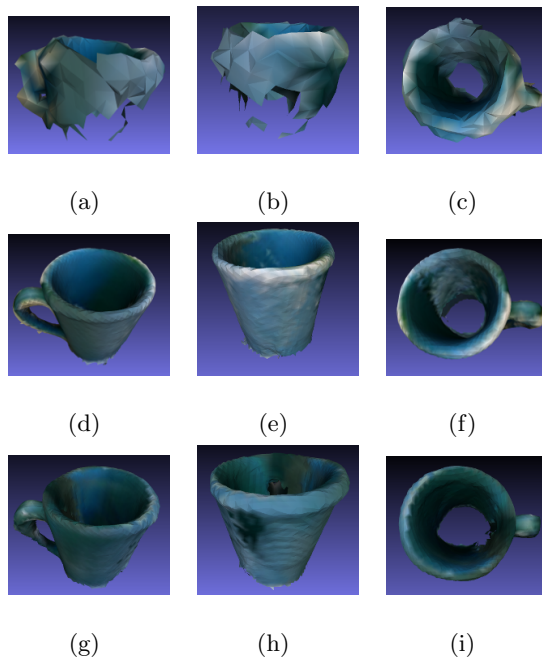


Figure 6.6: three models of a blue cup made by processing the same scenes with different modeling options. Each column shows all models from approximately the same viewpoint. First row: 1-cm map resolution. Second row: 3-mm map resolution. Third row: 3-mm map resolution and improved per-pixel color confidence weighting when adding frames to the map.

new “background” map containing exactly the contents of this scene, then run the scene of fig. 6.3(d) through this procedure to end up with the background map shown in fig. 6.3(f). The two scenes are represented here by the first frame of each video; each scene is hundreds of frames. The areas behind the object in each scene appear in the background map. While processing the second scene, we also find and model the object of fig. 6.3(e), which is in the first scene but not the second, and that of fig. 6.3(g), which is in the second scene but not the first. These are in fact the same object in different positions, and it is possible to discover this information.

This system does not yet include a component to do *object discovery*, the clustering of views of objects in different scenes into groups corresponding to physical objects. However, we intend to integrate the object discovery work discussed in ch. 8 with this system. Since our object discovery relies on high-accuracy models, we want to model all objects in as



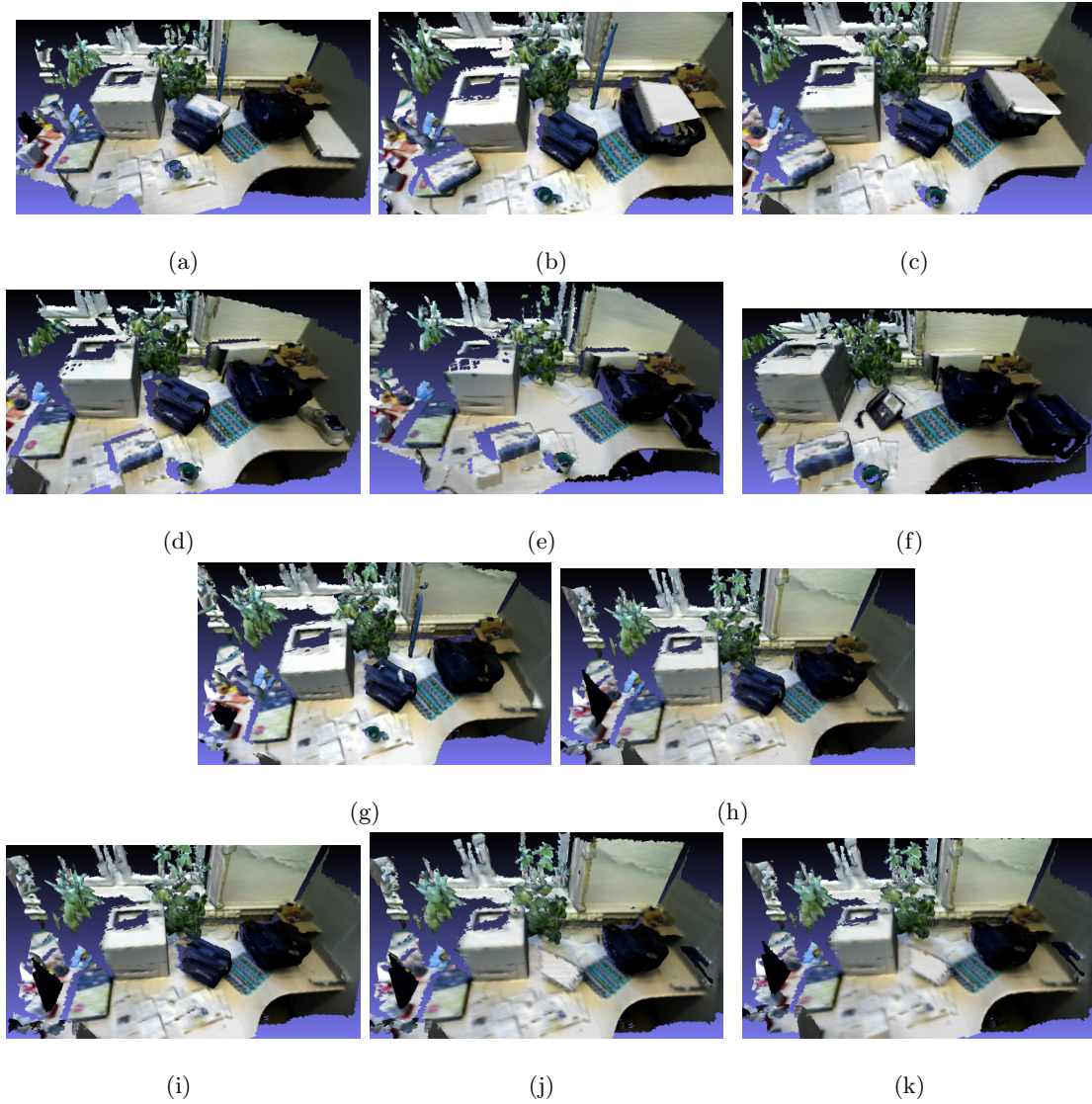


Figure 6.7: background maps made from a sequence of scenes. (a) - (f) six scenes of an office environment encountered sequentially. (g) - (k) the background maps obtained after processing 2, 3, 4, 5 and all 6 scenes, respectively. See fig. 6.8 for the objects segmented from these scenes.

much detail as possible. For any algorithm that maintains a large number of maps, memory is a bottleneck. Since we have approximate segmentation information for objects, we can restrict the sizes of the maps we use to represent them. Because these maps are so much smaller than the full scene, we can afford to make them much higher-resolution than the

scene map, allowing us to manipulate many object models in video memory at once and decrease runtime.

Each of six scenes of an office was run through our system in order. The scenes, and the background maps obtained in sequence, are shown in fig. 6.7. All object models obtained are shown in fig. 6.8. We have used these results for a comparison with the batch object segmentation method of [77] (see ch. 8). There are 36 unique object views in these six scenes. The batch method finds each of the 36 and creates an additional 45 non-object models, for a precision of 40%. The method of this chapter creates models for 24 of the 25 views that it should find (an object that doesn't move until scene 4 of 6, for example, can't also be segmented in scenes 1 and 2 with an online method); it also creates 12 duplicate models and 38 non-object models, giving a precision of 32% if duplicates are considered false positives or 49% if they are considered true positives. Images of all the models found with online segmentation are also available online; see sec. 6.8 for a URL. The objects found with off-line segmentation are shown in fig. 6.9 as the non-grayed-out parts of the maps of those scenes.

We can control the visual quality of object models in a variety of ways. The first two rows of fig. 6.6 show the effect of changing the resolution of object models. The third row demonstrates a weighting scheme for adding information to the map in which the colors of pixels in the frame are downweighted according to their distance from depth discontinuities in the frame when being added to the map. This downweighting alleviates problems caused by depth and color image misalignment in the input RGB-D frames, which occurs even when the camera is stationary due to sensor noise and imperfect calibration. The tradeoff we observe is that the more accurate the model color, the less accurate the shape. The models in fig. 6.6 were generated from two videos, one of an empty surface and one of the surface with the cup on it, in which the camera viewed the object from almost all viewpoints, demonstrating that even using a SLAM algorithm without loop closure, we can make a complete and accurate object model. (The Patch Volumes framework includes loop closure, but it optimizes constraints between volumes rather than between frames, as our use case would require.)

As mentioned before, we have the option of using an empty map as the initial background map when processing a scene. One interesting effect of using this option is that we can process non-static scenes. If an object in the scene disappears *while the camera*



*is pointing elsewhere*, then when the object’s previous location is once again in the field of view we can detect it as moved and build a model of it; similarly if an object appears while the camera points elsewhere we can model it once we see it. An example is given in a video at <http://www.cs.washington.edu/research-projects/robotics/rgbd-segmentation-online/>, and representative frames from this video are shown in fig. 6.10. In this mode, rather than differencing against a map created only from previous scenes, we difference against the current background map at each frame. Because it takes many frames for surface locations to change in volumetric maps, an object that appears or disappears must be viewed in many frames for enough evidence to accumulate in the map for previously present surfaces to be overwritten.

One limitation of online change detection is that if an object appears in the scene but partly outside the volume enclosed by the map made so far, there is no spatially localized information about change in the part of it that is outside the map. To build a complete model of such an object, it is necessary to spatially regularize change detection results and guess at object extent. This is why we emphasize including uncertainty information in our object models by using a representation that can encode free space or unseen space around the boundaries of the previously seen part of an object: this can be thought of as telling us in what directions it is possible for the object to be larger than the portion of it that we have seen so far.

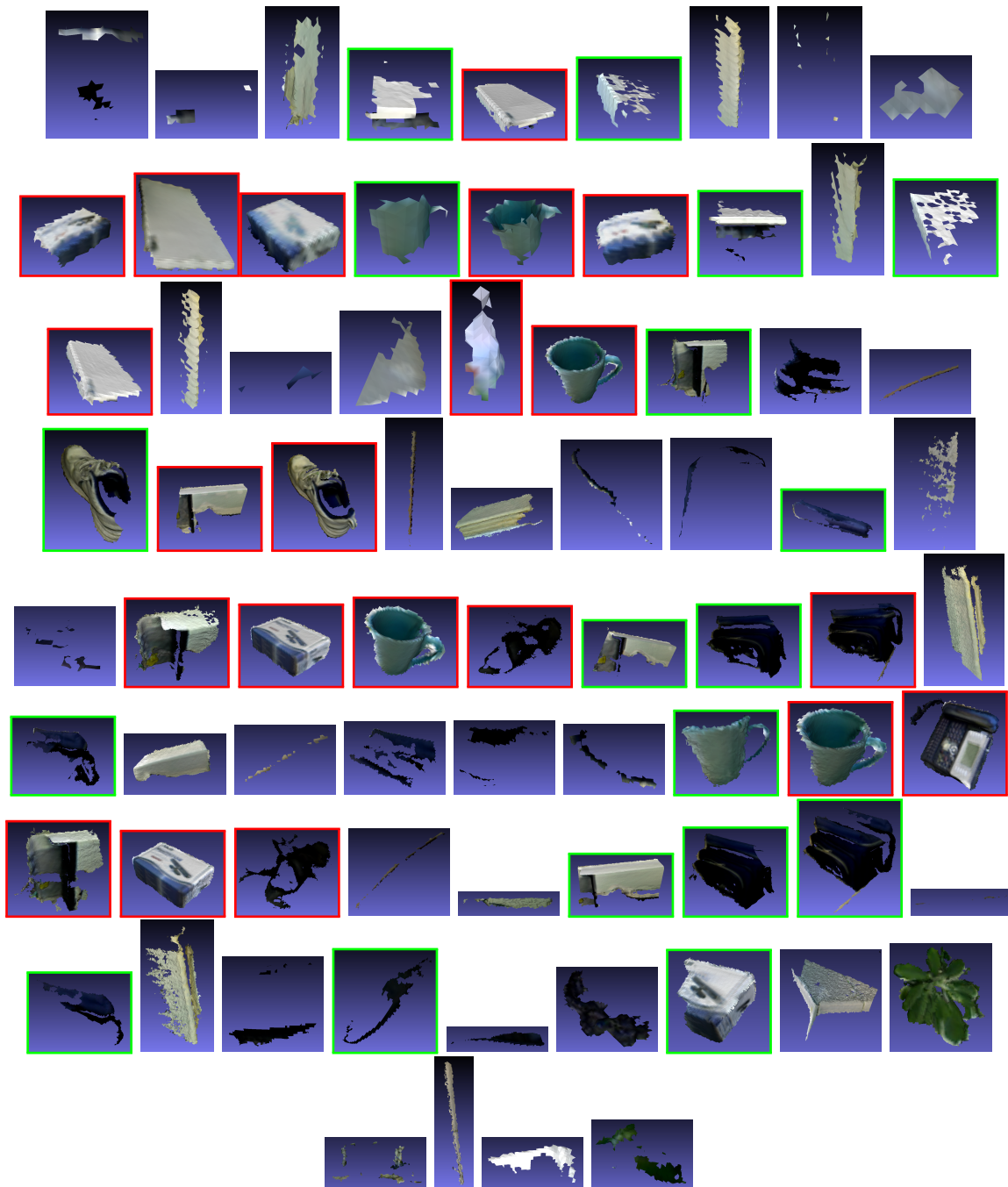


Figure 6.8: The objects segmented from the scenes of fig. 6.7. Segments representing actual objects are outlined in red; segments that duplicate existing objects are outlined in green; the rest are non-objects and should not have been segmented. Among actual objects, we see views of a large textureless box, a textured box, a cup, a phone, a lunchbox, a clearly visible shoe and the shadowed shoe from those scenes where it is under the desk (lower right corner of scene).

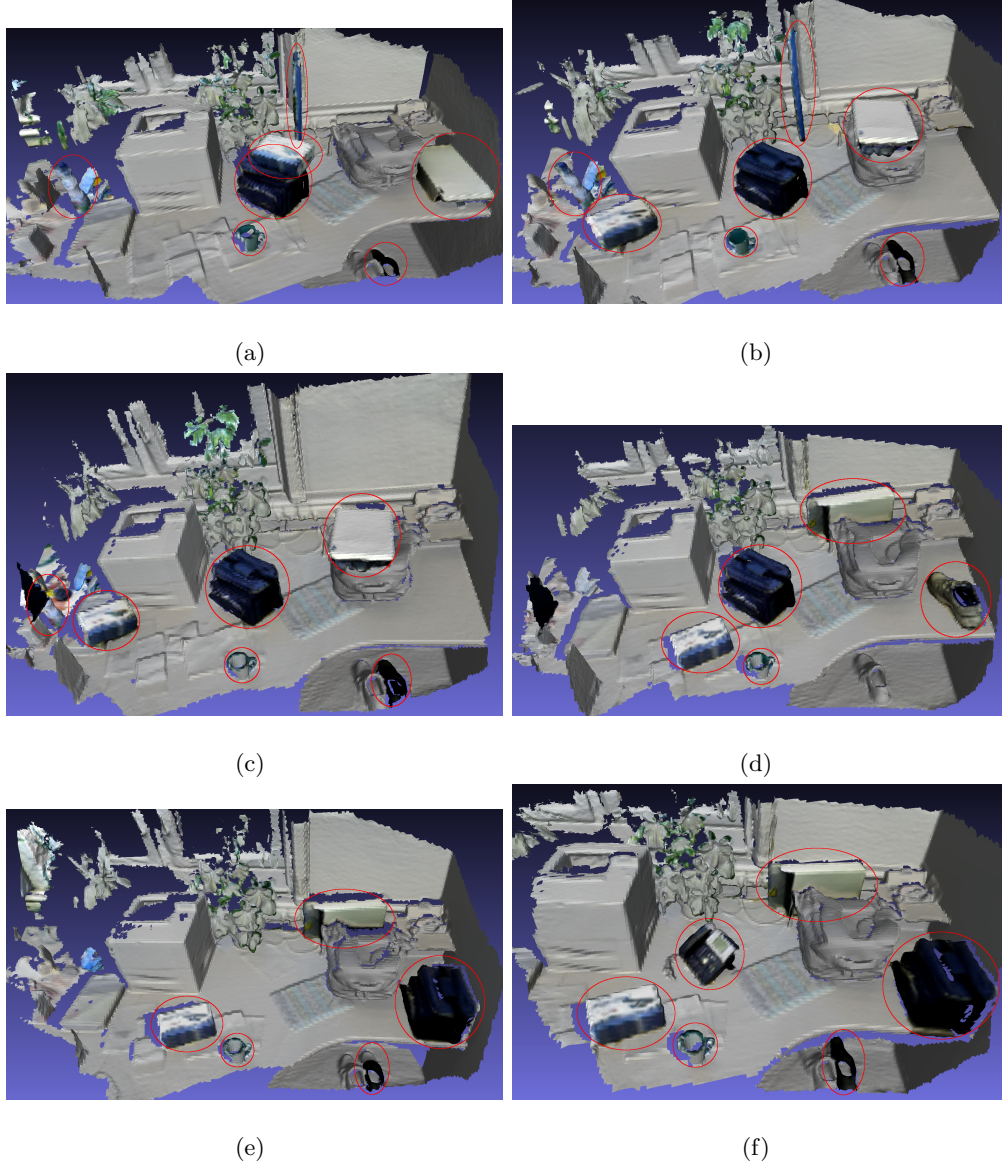


Figure 6.9: visualizations of the segmentations produced by the off-line method of ch. 8 for the six scenes shown in fig. 6.7. Non-grayed-out areas are segmented objects. Those segments that are correctly identified moved objects are also circled in red.

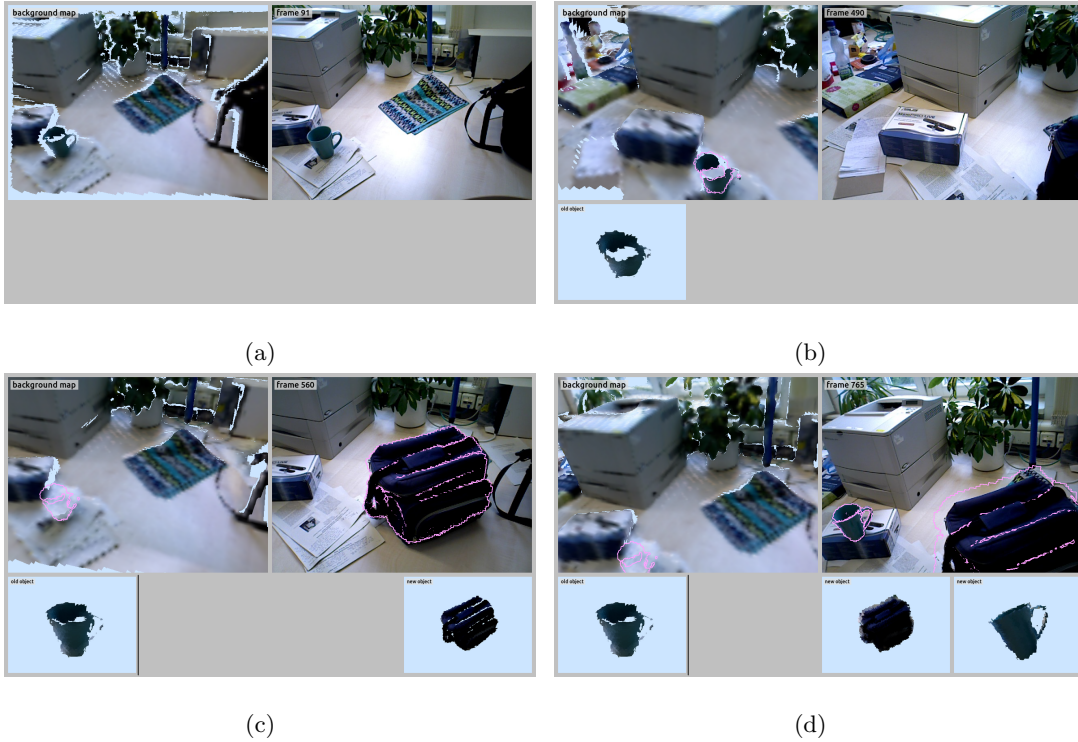


Figure 6.10: four frames, in chronological order, from a video made to visualize online modeling of a single scene. In the recording used here, the camera pans left, then right across a desk. A cup seen near the beginning moves while the camera is off to the left (between (a) and (b)), and at the same time a lunchbox appears at the right of the scene. At the end of the video there are therefore three object models: one of the cup in its old location, one of the cup in its new location, and one of the lunchbox made once it appears. The background map contains the rest of the scene, which does not change during recording. In each frame, the background map is shown on the left, the current frame is shown on the right, and object models created so far are shown at bottom. Locations of the object models in the background map and current frame are also highlighted in pink.

The outlines of object models in the video drift from the true object boundaries over time as each raw frame introduces more misalignment between color and depth edges. Thus the boundaries shown here in (d) look less accurate than those in (c).

## 6.6 *Other Sources of Segmentation Information*

We are not limited to acquiring segmentation information from change detection. Any source that provides segmentation information in a similar format can be used. We need segmentation results given in the image plane of some camera (i.e. 2-D segmentation results over a 3-D scene) and they must not include uncertainty information (such as that shown in sec. 4.5), since we cannot accumulate such information in volumetric object models. In particular, the human object demonstrations of sec. 5.4 are well designed for use with online object modeling. It would also be possible to use a system in which a robot manipulates the scene without knowing what objects are present, as is discussed in sec. 7.3, to cause motion from which segmentations can be derived to provide segmentation information. As discussed in sec. 4.5, at present such a scheme is too hard to use due to the complexity of obtaining segmentations from non-informed motion. However, this is a good avenue for future work.

## 6.7 *Efficiency*

Our system is intended for interactive use, so efficiency is a concern. Runtime depends on the number of object models that must be processed at each frame. Currently, when the number of models is under twenty, processing a frame usually takes .7 to 2 seconds. More than half of this is in change detection, the only major component of the system that does not run on the GPU. Change detection can be made to work on the GPU, and has not to date due mainly to time constraints. Our scene flow, which we use in analyzing segmentation demonstrations, runs on the GPU. It takes one to five seconds per frame pair, depending on scene content: the less visual texture the scene has, the more iterations of smoothing are required.

## 6.8 *Summary and Future Work*

We have shown fast and accurate reconstruction of multiple rigid objects online during scene reconstruction. This allows us to represent a scene with one 3-D model of the static background as well as a model of each object we ever observe to have moved. Object models are updated online, and we can direct the camera’s motion to improve model quality. Reconstruction uncertainty is encoded in volumetric maps of movable objects and the back-

ground. We can incorporate object segmentation information from change detection over multiple scenes as well as from other sources.

There are many avenues for future work with volumetric maps and online object segmentation. For example, we have yet to fully integrate the object demonstration system discussed above with the active object modeling system, which will in general involve splitting and merging of object models after completing a demonstration. Also, since volumetric maps store some notion of depth uncertainty, we plan to read such information out of them during rendering, to more accurately represent model uncertainty for change detection. The object models acquired with active modeling also will be used for object discovery so that we can track objects over time.

We intend to integrate information about segmentation from the sources discussed here with other sources such as 1) detection of previously learned partial or full object models, using techniques similar to those of [111] but with our high-accuracy sensor model, and 2) active exploration by a robot. The latter source of information is very nondeterministic due to the difficulty of causing *useful* motion when starting with very little knowledge, and will require more probabilistic segmentation techniques than we use in this work.

Additional images can be found at <http://www.cs.washington.edu/research-projects/robotics/rgbd-segmentation-online/>.

Robust online segmentation and object modeling is a prerequisite for tracking previously unknown objects, allowing object-level understanding of, for example, egocentric video [1] involving textureless objects. For tracking partially built object models through such video we might use dense alignment methods based on those we use in this work, change-detection-based methods similar to [111], or modifications of depth-only methods involving point matching between frames (e.g. [144, 159]).



## Chapter 7

## ACTIVE AND INTERACTIVE OBJECT SEGMENTATION

## 7.1 Introduction

All the segmentation methods discussed so far have been *passive* segmentation techniques: we are given a set of measurements from which to extract a segmentation. We should be able to improve results if we have access to a physical agent with the ability to rearrange the scene to give itself more information, because future measurements can be directly related to a notion of what information is contained in past measurements. If the robot's camera is movable, moving it can provide views that would never be obtained by a passive agent: humans who might otherwise move the camera probably won't know the robot's intentions or current knowledge, and it is very unlikely that humans would feel the need to rearrange objects for the viewing benefit of a camera with a fixed position in the environment, as they would be unaware that this would be helpful to vision algorithms. Additionally, some objects are moved by humans, but very seldom, so that a robot operating in an environment might not learn all objects for months or years. These arguments motivate *active object segmentation*, in which the robot literally takes matters into its own hands and causes the camera or objects to move. (The more general concept of *active perception* was introduced by Bajcsy [10].) Having the ability to choose what data to process means a robot can segment objects when it wants and can choose how much of a scene to understand before moving on. Importantly for algorithms using formal methods in robotics (e.g. [112]), it is also possible to control the amount of certainty we have about object boundaries: the robot can manipulate an object whose configuration it is unsure of until its model has the desired amount of uncertainty.

In this chapter we present two pieces of work in progress on active segmentation. The first is a method for choosing motions of a camera mounted on a robot to ensure that the robot continually sees new parts of the scene. We can incorporate a wide range of preferences for selecting viewing locations so as, for example, to focus on completing our models of regions we know to be objects (e.g., from change detection) but that are partly

unseen.

The second project, less developed as yet, involves using the robot to cause objects in the scene to move so that we can use the resulting motion cues. There is a large body of existing research on causing object motion in order to segment objects—which is sometimes referred to as *interactive segmentation*—so there also exist methods for choosing what manipulations to use to cause object motion [32, 14, 57, 99, 96, 13, 176, 98, 80]. One current challenge in this work is selecting what manipulations to make; the only one of these papers that chooses manipulations in an informed way is [32], which segments a pile of objects one at a time and uses a termination condition based on determining that the robot has separated one object from the rest. Therefore our goal in this project is to find an intelligent way to select manipulations.

## 7.2 Active Segmentation: Causing Camera Motion

The goal of active vision is to change intrinsic or extrinsic properties of the sensor in order to get new and improved information. In the case of 3-D object modeling from vision, this can mean moving the camera to view the scene from a new angle or to get a close-up view of an object that has been localized and is being modeled. We discuss using active vision for map completion and for segmentation.

### 7.2.1 Introduction

In active segmentation we make use of the fact that our robot has a movable camera. The setup we use in this section has the camera attached to a simulated free-flying robot similar to a quadrucopter or to the WAM arm used in previous chapters. The goal of this work is to aid 3-D segmentation, given 2-D segmentation tools such as we have discussed in previous chapters, by viewing previously unseen parts of the map. The *map completion* problem is defined as moving the camera to see all surfaces in the map, and the (less well-defined) *active segmentation* problem is to move the camera to improve our segmentation estimates for objects.

For our purposes, map completion is a proxy problem for object model completion. Consider the online mapping and modeling system discussed in ch. 6. If an occluder is removed and we observe an object behind it, change detection alone cannot segment the



object completely because there is no change detection evidence for the section of the unoccluded object that was behind the occluder. However, since we know the boundaries of the occluded region (which includes the object as well as some other surfaces), we can still make a model that contains the object by taking advantage of having a camera on the robot. The object and background in the resulting model cannot be separated perfectly until we see the same region of background without the object. Therefore we wish to model at least some interesting portions of each scene we see as completely as possible in order to give us maximal information for processing future scenes. Another reason to use active vision for object modeling is that the more complete an object model, the more useful it is for object discovery.

The most popular paradigms for map completion (sometimes called “exploration”) are frontier-based and next-best-view exploration. Given a map representation that divides space into free, surface, and unseen areas, *frontiers* are defined as the parts of the map that border both free and unseen space [195]. We use the volumetric map representation introduced in sec. 2.2. Frontier-based approaches to map completion iteratively select frontiers to view with the sensor (usually a camera). Next-best-view approaches, which are a type of generate-and-test algorithm, select a set of possible future viewpoints to score according to some notion of usefulness of a view, which depends on the current contents of the map. Notions of usefulness on a volumetric map are generally computationally expensive to calculate.

We introduce a frontier-based framework for solving map completion and related problems such as active segmentation. We do this with a general representation of preferences on what areas are viewed next. We use a computationally inexpensive method for selecting views given preferences, allowing us to make decisions at a higher rate than any other 3-D next-best-view approach of which we are aware.

### 7.2.2 Related Work

Every next-best-view view selection algorithm proposes a set of possible sensor poses for the next sensor reading, selects the best by some metric (usually based on information gain), moves the sensor to the selected pose, and integrates the resulting reading into a representation of the scene. Information gain is very hard to define for applications in

which the goal is to measure variables that are currently completely unknown, so most work uses rough approximations for view scoring. Work on selecting next best views by using projections of unseen space into potential camera poses goes back at least to 1985 [40], and there has been a large amount of work recently using similar algorithms in 3-D, e.g. [109, 174, 140]. Thrun et al. [171] discuss robot exploration during 2-D mapping by selecting next views to maximize an approximation of information gain. We compare to such an approach in 3-D.

The most common variant is *myopic* next-best-view planning, in which only a single future view is considered. For example, Triebel et al. [174] select a next pose for a laser range finder to maximize an approximation to information gain given by assuming a point distribution over readings at each single scan (in particular, they assume the first voxel along a ray with occupancy probability over a threshold will be hit by the beam). Their pose proposal step constrains next views to be reachable by the manipulator on which their sensor is mounted. Gonzalez-Baños and Latombe [64] select a next view using a combination of traversal cost and a novelty score given by the total volume of the currently unseen area that would be seen by the new view. Potthast et al. [140] use the penetration depth of each hypothetical pixel ray into an unseen region as an estimate for information gain, reasoning that the more unseen space could be seen through, the more likely we are to find out where the surface actually is.

By contrast to these myopic methods, Atanasov et al. [8] select camera poses for table-top object classification by planning a sequence of two or more viewpoints for each pre-segmented object. They maintain a discrete distribution over object models in a database to estimate which known object is currently being observed. They include traversal cost in planning. Similarly, Hollinger et al. [81] first select a set of overlapping views of a ship hull and plan a path that can use any subset of these views to cover the entire hull for visual search purposes. They use this method only after building a complete object model; they use view selection to reduce uncertainty in that model. In both these projects, view selection and path planning are done off line.

Blodow et al. [19] select next-view poses for a mobile robot with a laser scanner by ray-tracing in 2-D (similar to a 2-D laser scanner such as a Hokuyo) at each proposed pose until each ray hits a non-free voxel. Their score function rewards poses from which the numbers of frontier voxels (representing new information) and surface voxels (in order to

perform alignment to the existing map) thus rendered are equal. This is an example of a technique that fits in both the next-best-view and frontier-based categories.

Holz et al. [82] quantitatively compare frontier-based and non-frontier-based view selection methods and find that the two frameworks yield similar results, despite the fact that next-best-view approaches consider more information in view scoring than the relatively simple frontier-based methods they test. Their main research contribution is a quantitative study of early stopping in map completion, in which replanning happens as soon as the current target map region (part of the frontier) is visible rather than after the current target pose is reached.

Shen et al. [160] use a gas diffusion model to identify frontiers in a low-memory, low-computation setting and show that they can produce a map of similar quality to one produced by a more conventional frontier-based algorithm operating on a non-embedded platform.

The theory of submodular optimization (an analogue of convexity for discrete optimization) allows for efficient optimization of objectives in which acquiring the same or similar information loses value over time as the variables involved are modeled better and better by the growing set of previous views. Golovin and Krause [63] define *adaptive submodularity* for problems in which action selection should depend on recently acquired information. They present a greedy algorithm for optimization of adaptive submodular objectives, which is applicable, for example, to the map completion problem. Javdani et al. [91] note that information gain is not adaptive submodular, and so is theoretically not usable as an objective for view selection. They suggest instead using objectives based on hypothesis pruning, in which actions are selected to reduce the set of plausible hypotheses by the maximum number of elements at each timestep. They use an objective based on hypothesis pruning with a tactile sensor to do object pose estimation.

One issue in designing map completion-like algorithms is selecting a termination condition. Object model completion using probabilistic measures of completeness as a function of viewing direction is demonstrated by Zillich et al. [198]. They use point feature matches to a model as the basic unit of information about an object, and update the model after each view. One of the nicest features of their work is a condition for when to stop acquiring new views of an object based on the distribution of point features in its model. Krainin et al. [109] use a similar measure of model completeness defined at all vertices of an object

mesh, which is assumed to be complete and watertight before view planning.

Our setup is different from those in most of the references above (excepting Zillich et al. [198]) in that they all use sensors mounted in fixed positions on a robot base, whereas we attach a camera to the end of a robot arm. Our scenario is more general than that of Zillich et al. in that they are interested in modeling a single object with a known background, whereas we are interested in modeling the entire scene, starting with no knowledge of any surfaces in the scene. Therefore we integrate camera motion with a SLAM system. We use an online SLAM system [73] and we desire to map the scene as quickly as possible, without spending valuable camera time sitting in place during slow planning. Our motivation for this decision is the intuition that the more of the time the camera is in motion, the more quickly we will acquire new information. Since our system is designed for use on a robot, we want the entire system to run as close to in real time as possible. This motivates the use of a novel frontier-based approach in which we select views using a simple scoring method and expect to move only partway to the selected view before selecting another.

Our ultimate interest is in using view selection for object segmentation. Possibly the most similar existing work to this goal is that of Holz et al. [83], who use view selection for object detection in a bin picking task. They use a pipeline of geometric primitive fitting followed by object detection. Each voxel in a volumetric map is given a value for view selection purposes; values are based on the outputs of the two pipeline stages at each voxel, which can change over time as view selection and the detection pipeline are run alternately. Important differences from our work are that Holz et al. detect a known object rather than novel ones and that they assume their object’s shape is composed of a small set of geometric primitives.

### 7.2.3 Framework Overview

We have implemented a view selection and motion planning framework as an extension to the online 3-D reconstruction and object segmentation system introduced in ch. 6. We show a flow chart of modules in the entire system in fig. 7.1. We run mapping in one thread, object modeling (introduced in ch. 6) in another, view selection and motion planning in a third, and motion execution also in its own thread. View selection uses the output of mapping, and produces a control input for a (simulated or real) robot with a movable camera. For

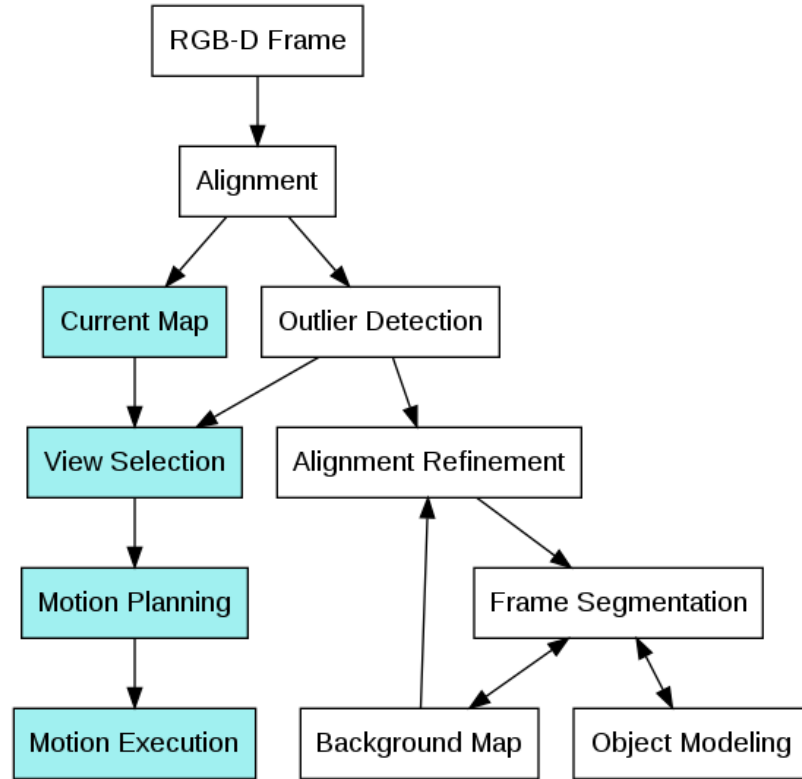


Figure 7.1: a diagram of our active online mapping and object segmentation system. Blue nodes are not present in the system of ch. 6. This diagram should be interpreted as showing information flow rather than execution flow, as various parts of the system run in parallel.

example, the camera can be attached to a kinematic-chain manipulator or can be free-flying (this is one approach to simulating something similar to a quadrucopter). Automated view selection could also be used as a hint to a human carrying the camera, similarly to [51], in which the user is shown what areas of a map being built are still incomplete.

Active mapping involves 1) view selection, which suggests future camera poses, and 2) motion planning given a target camera pose. Alg. 7.4 outlines our active mapping update algorithm. After initialization of active mapping, this update is run repeatedly until an externally defined completion condition is met.

At any given time there can be a view target (a location on a known surface or on the 3-D frontier) defined. (Initially there isn't.) We plan to move toward a configuration viewing

---

**Algorithm 7.4** pseudocode for our view selection and motion planning algorithm. This code runs iteratively starting as soon as the map is initialized and ending once a termination criterion is reached.

---

**Input:** map  $\mathcal{V}$

---

```

1: if  $ViewTarget$  defined then
2:    $Success \leftarrow (\neg TARGETINFreespace(ViewTarget, \mathcal{V}) \wedge \neg TARGETUNMEASURABLE(ViewTarget))$ 
3:   if  $Success$  then
4:      $(Plan, Success) \leftarrow TRYPLANToPOSE(p, \mathcal{V})$ 
5:   if  $Success$  then
6:     execute  $Plan$ 
7:   else
8:      $ViewTarget \leftarrow \text{undefined}$ 
9: if  $ViewTarget$  not defined then
10:   $\mathcal{M} \leftarrow EXTRACTMESH(\mathcal{V}) \quad \triangleright \text{Marching Cubes}$ 
11:   $\mathcal{F} \leftarrow EXTRACTFRONTIER(\mathcal{M})$ 
12:   $\mathcal{V} : \mathcal{F} \mapsto \mathbb{R} \leftarrow VALUATEFRONTIERTRIS(\mathcal{F})$ 
13:  sort  $\mathcal{F}$  by  $\mathcal{V} \quad \triangleright \text{index into } \mathcal{F}$ 
14:   $i \leftarrow 0$ 
15:   $Success \leftarrow \text{false}$ 
16:  while  $\neg Success$  and  $i < |\mathcal{F}|$  do
17:     $(T_{next}, i) \leftarrow NEXTTRIByVALUE(\mathcal{F}, \mathcal{V}, i)$ 
18:     $\mathcal{P} \leftarrow SUGGESTVIEWINGPOSES(T_{next})$ 
19:     $\mathcal{P} \leftarrow FILTERBySTATICREACHABILITY(T_{next}, \mathcal{P}, \mathcal{V})$ 
20:     $\mathcal{P} \leftarrow FILTERByCOLLISIONCHECKING(T_{next}, \mathcal{P}, \mathcal{V})$ 
21:     $\mathcal{P} \leftarrow FILTERByNONVISIBILITY(T_{next}, \mathcal{P}, \mathcal{V})$ 
22:     $\mathcal{P} \leftarrow FILTERByNONMEASURABILITY(T_{next}, \mathcal{P}, \mathcal{V})$ 
23:    for all  $p \in \mathcal{P}$  do
24:       $(Plan, Success) \leftarrow TRYPLANToPOSE(p, \mathcal{V})$ 
25:      if  $Success$  then
26:        execute  $Plan$ 
27:      end forall
28: end the experiment if we exhausted the pose suggester (there's nothing unseen that we can move
    the camera to)

```

---

the current target if there is a target. We choose a new target when the target becomes seen (i.e. moves off the frontier into known-free space) or is found to be unmeasurable. (In real scenes, some surfaces will never or rarely return readings to an infrared sensor. We model this phenomenon by quantizing view targets, remembering which quantized targets were viewed but did not return valid depths, and avoiding targeting these views again.)

Target selection starts with EXTRACTMESH, which runs the Marching Cubes [120] mesher for volumetric maps. Our version extracts both the zero level set of the signed distance function and the boundaries between empty and unseen space (frontiers), and we use EXTRACTFRONTIER to enumerate only the frontier triangles in this triangulated mesh. VALUATEFRONTIERTRIS assigns a viewing utility to each triangle; we will use these values to rank views later. We run through triangles in decreasing order of utility, suggesting one or more poses from which to view each in turn and attempting to plan to each of these before giving up on a triangle. Viewing poses are suggested using only information about the single triangle in question, so we need to filter them (lines 19–22 of alg. 7.4) based on reachability given physical robot constraints, collision checking with the environment, visibility of the target triangle from the suggested camera pose (again, there can be obstacles), and information about unmeasurable surfaces (in real scenes, some surfaces don’t return readings). When planning (line 24) is successful, we set the triangle’s centroid as the new view target.

Having a long list of potential view targets (a room-sized map in progress with a 2-cm voxel resolution contains between 100,000 and 1,000,000 frontier triangles) gives us robustness to failure of planning to a single target. At any point, if there is *any* frontier region we are able to plan toward, we can continue mapping.

#### 7.2.4 Simple and Inexpensive Planning

In most previous work, view selection has been a computationally expensive operation run once each time the robot reaches a new target. We make view selection/planning inexpensive enough to be run continually while the robot is moving. This means that we might plan the robot’s motion for, for example, time interval  $t \in [1, 2)$  when  $t = 0$  and we don’t yet have all the map information we will at  $t = 1$ . However, because our planning is fast and we plan short motions, we have *almost* the most recent information, and we have the advantage that

the robot is continually improving the map rather than sitting idle during planning. One consequence of this approach is that Holz et al. [82]’s “rapid rechecking” isn’t so important for us since we don’t move the camera very far at a time. Holz et al. check at each processing iteration whether the current view target has already moved into free space, and if so cancel the rest of the movement toward it; in our case, this cancelation consists of setting the view target to “none”. The rechecking strategy is, of course, still useful: on the map completion problem in a simulated environment that we will discuss shortly, rechecking saves us 15% of travel distance and 35% of runtime. All the results we report in this section use rechecking.

To help ensure that the camera will move continuously, we keep a queue of up to  $k = 2$  planned future motions pending at all times. The queue size limit helps avoid planning with too much stale information.

Another motivation for this continual-replanning approach is that the faster our replanning is, the sooner it can make use of new information in the environment; in the limit, if planning can run once per incoming frame, it becomes possible to replan based on objects that move during planning. This will allow the camera to follow objects that are moving as they are being observed, in order to complete object models *during* the movement of those objects, or to track how nonrigid objects’ shapes change as they move.

One benefit of continual replanning is the ability to use simple and fast planners. Common options for motion planning of general-case motions are rapidly exploring random trees [115] and iterative path smoothers such as CHOMP [199]. A smoother must be initialized with an approximate path (which can be very simple, such as a straight-line path in configuration space). We have experimented with both methods. RRTs as implemented in OpenRAVE [49] run for two to four seconds on our problems (with a large amount of occlusion in the environment), and CHOMP usually runs for .5 to 1 second. Due to our continual-replanning approach, however, we have another option: configuration interpolation. After selecting an end pose for the path, for kinematic chains we use linear interpolation in configuration space, and for a free-flying camera, whose configuration is its pose, we use linear interpolation of translation and spherical linear interpolation of rotation. This planner runs in an average of .03 seconds, with a standard deviation of .02. Linear interpolation is a reasonable choice because we keep paths short: in order to avoid having the execution of a path take much longer than planning the next path, we execute only the beginning of each planned path. Thus each view target takes one or more executed plans to reach. In return



for the advantage of being able to use very simple planning, we have the potential disadvantage of not being able to move between two very separated areas of the environment that are both interesting; that in general requires more sophisticated planners. As the emphasis of this work is on simplicity and on planning only to nearby locations, we leave exploration of this tradeoff for future work.

### 7.2.5 View Selection

The mesh extraction algorithm in Patch Volumes is a version of marching cubes modified to provide meshes for both surfaces and the frontier. We use the centroid and normal of each frontier triangle as a potential view target location and orientation. We limit the size of this set of triangles by subsampling with a required minimum distance between samples.

Function `VALUATEFRONTIERTRIS` in alg. 7.4 assigns a value to each frontier triangle according to some idea of how useful it would be for the camera to view that triangle next. Where the camera might view the triangle from depends on `SUGGESTVIEWINGPOSES`, but almost certainly there will be more than one triangle in the view, so the value of a triangle is allowed to take information about other parts of the scene into account. Our framework for view selection is general and the particular algorithm can be anywhere on the efficiency spectrum, from a simple approach that values all frontiers the same to a sophisticated algorithm that uses globally defined information about surfaces in the scene. For example, one of the most common approaches to scoring views is to render the scene into each view and process the set of scenels seen by each view. We evaluate such an approach later in this section.

The simplest goal for a view selection algorithm is map completion: selection of new poses can cease when there is no unseen space left in the reconstructed map. Some value functions defined to achieve map completion are given in sec. 7.2.6. We define a framework for selecting views rather than a single algorithm because we are also interested in more complex goals. The effects we can achieve with our per-scenel valuation include:

- Sometimes (e.g., with the use of change detection) we have a partial segmentation of an object and can perform the rest of the segmentation if we choose views to see the edge of the surface region known to be part of the object. By assigning high value to frontier triangles near interesting surfaces, we can force the system to view unseen

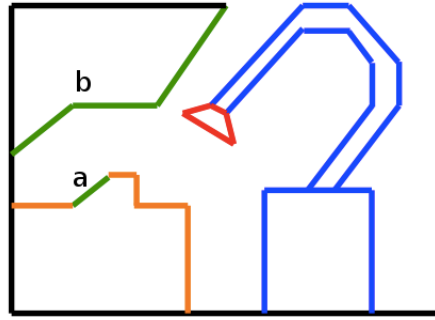


Figure 7.2: an illustration of potential motions being blocked by unseen space. The robot (blue) is holding the camera (red). Black shows some of the edges of the map volume. The camera has already seen a table with an object on it (orange). Green shows frontiers: edges between free and unseen space. The robot would like to move the camera to view the left side of the object (a), which is currently on the frontier, but due to the physical extent of the robot, it would have to penetrate the unseen space above the table (b) in order to do so. This problem suggests having a way to prioritize looking at unseen space far from known objects. (Any algorithm in our framework would still look at the blocking space in this figure eventually, because all suggested view targets are considered in order; the issue is one of efficiency rather than completeness.)

parts of known objects before moving to areas not already known to contain objects.

- One possibly desirable behavior when modeling objects is to view all sides of an object before moving to another. This objective function is very non-submodular and so cannot be achieved by submodular optimization methods, which have nice theoretical properties [91]. By defining a value function that rewards both proximity to the current camera position and proximity to partially modeled objects, we can achieve this kind of behavior.
- Especially when the robot is a kinematic chain, it is possible for there to be large regions of unseen space blocking the robot's path to an area it needs to reach. See fig. 7.2 for an illustration. In our framework it is possible to raise the value of frontiers in such regions over time so that seeing through them is given higher priority than trying to view unseen parts of the intended target region.

To guarantee that a view selection system will find *all* objects in a scene, map completion must be incorporated into the value function, because any unseen region of space can contain an object. However, prioritizing space by using partial information about known objects or about regions in which objects are likely to appear (as provided, e.g., by [9]) allows us to make the best use of our time if it is known that there is a time budget but the budget is unknown.

### 7.2.6 Experiments

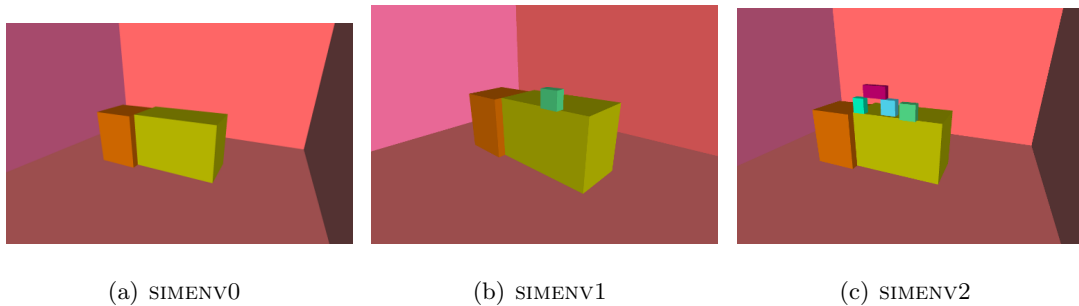


Figure 7.3: the three simulated environments SIMENV0/1/2 used in our simulation experiments. They vary only in the objects on the table.

We show three experiments: one to compare view selection algorithms, one to demonstrate the use of our framework for object completion, and one to explore differences between flying and arm-mounted cameras. Due to limited hardware availability, the experiments we present here are all in simulation. We use three similar simulated scenes, each containing a yellow and an orange table inside a room with walls, floor and ceiling that are varying shades of pink. One scene has no objects on the table, one has one object, and one has four. These scenes are shown in fig. 7.3. Each scene is bounded by walls so that it cannot take infinite time to complete a map. Each scene is the same size, about 4 m<sup>2</sup> horizontally and 2 m vertically. In two experiments we model a camera attached to a free-flying small spherical robot (an approximation to a quadrucopter). In the other, we model a camera mounted on a kinematic chain: a WAM arm [12] attached to a stationary base. Both models are shown in fig. 7.4.

All space in the map is initially unseen. To allow the camera to move at all, we carve

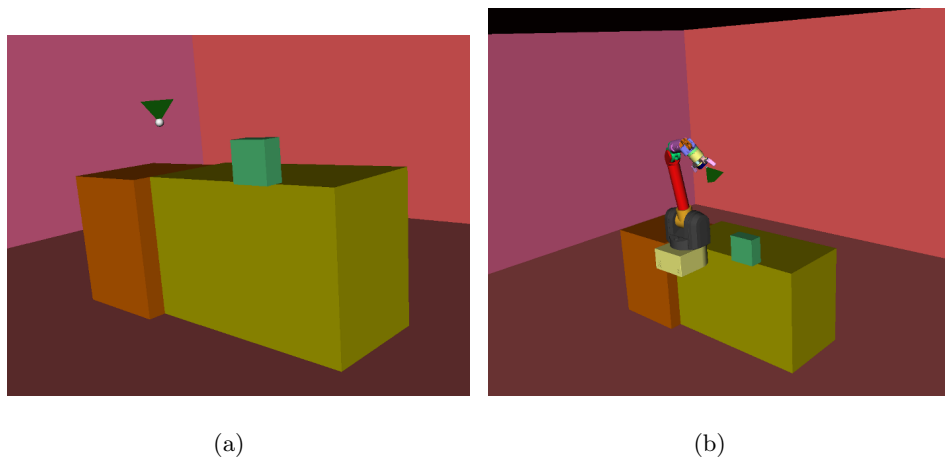


Figure 7.4: the two simulated robot models we use in our experiments. (a) a free-flying camera (technically mounted to a small white sphere); (b) a camera mounted on the palm of a WAM arm with a Barrett hand. We do not allow the base of the WAM to move. In each image the camera viewing direction is indicated with a green frustum.

out some “known” free space around the robot before starting mapping. In the case of a free-flying camera we provide a .4-meter cube of initial free space; in the case of the WAM assembly, we provide space including the entire robot and some extra space for the arm’s elbow to be able to bend in its first few movements.

When running with simulated environments, we provide Patch Volumes with ground-truth alignments rather than using its alignment functionality, because its alignment is unstable in environments with zero color or depth texture. We have verified empirically that adding a small amount of color noise to otherwise single-color-valued surfaces is enough to avoid this instability, but that tactic is unavailable in our current software framework, so for ease of implementation we avoid running alignment in simulation.

There are at least three types of termination condition we can use in map completion-related applications: we are done mapping when there are no frontiers left in the map, when we are no longer able to plan to any remaining frontier (due to limitations of a kinematic-chain robot or due to hidden surfaces in the map caused by inaccuracy in alignment), or when there are no high-value scenels left in our list of potential targets. We end an experiment when any of these conditions occurs.

In our map completion experiments, when online change detection functionality is not needed, we disable online change detection and object modeling to better evaluate view selection. The object completion experiment requires change detection to identify surfaces of interest. Therefore runtime numbers from these different experiments are not comparable.

### *Map Completion with a Free-Flying Camera*

As a first step toward active segmentation, we evaluate our system on the simpler problem of map completion in a simulated environment (SIMENV1). This will help us choose a view selection algorithm to modify for the purpose of active segmentation. We use a free-flying camera, whose robot configuration consists of its pose, represented as a quaternion and 3-D translation. We compare the performance of four view selection algorithms on this problem:

- RANDOM-FRONTIER assigns a random score in  $[0, 1]$  to each frontier triangle. Holz et al. [82] refer to this algorithm as “random frontier”.
- DISTANCE-TO-CAM assigns a score  $U$  based on the distance between the camera pose at the start of planning and the suggested camera pose  $X$  viewing each frontier triangle:

$$U_d(X) = e^{-D_p(X)}, \quad (7.1)$$

where  $D_p(X)$  gives the pose distance between the camera pose at the start of planning and camera pose  $X$ , calculated as the Euclidean distance between the translations in meters plus 0.5 times the angular distance between the rotations in radians. We refer to this pseudo-unit as “meter-equivalents”.

We could equally well get the distance numbers into the  $[0, 1]$  range by dividing by the maximum distance observed rather than by exponentiating. Either would result in the same ranking of scores.

- INFO-GAIN assigns a score based on an approximation to the expected information gain of the suggested next view over variables representing the state of each voxel that would be seen. The current scene, including frontiers, is rendered from each suggested view  $X$  and the number of pixels  $N(X)$  in the hypothetical image that

would see currently unseen space is counted. The score is then

$$U_{ig}(X) = \frac{N(X)}{\max_{X \in \mathcal{X}} N(X)}, \quad (7.2)$$

where  $\mathcal{X}$  is the set of camera poses being evaluated. This is similar to the evaluation function of Blodow et al. [19], with the difference that since we update the map at each frame, we have no need to include a measure of map overlap in the evaluation function. The other main difference between these two algorithms is that we don't sample our suggested camera poses randomly.

- TRAVERSAL-COST measures the approximate traversal cost for the camera. The score of suggested pose  $X$  is

$$U_{tr}(X) = e^{-D_t(X)}, \quad (7.3)$$

where  $D_t(X)$  is the approximate traversal cost from the current camera pose to camera pose  $X$ . Distance is calculated as for DISTANCE-TO-CAM. We approximate the traversal cost similarly to how Wettach et al. [187] do: we create a graph whose nodes are the centers of a voxel grid overlaid on the map (for speed, we use a grid with larger resolution than that used for mapping) as well as the current and all suggested future camera positions. We then compute shortest paths between all pairs of nodes. The distance function we use between nodes takes both translational and rotational distance into account. (We arbitrarily assign the same rotation as that of the current camera pose to each node created from an arbitrary voxel center.) For view selection, we only need to use the distances between the current and each proposed future camera pose. Wettach et al., who want to move the robot along the resulting path, additionally use a continuous optimization to produce a smooth path; we compute paths to use in a separate step, so do not need to smooth paths here.

Due to the exponential dependence of the number of graph nodes in this approach on the configuration space dimension, we will probably need to explore an approach based on a probabilistic roadmap (PRM) once we move to higher dimensions (e.g. for our 7-dof WAM arm). In three dimensions we find this approach faster than a PRM.

- INFO-TRAVERSAL combines the information gain and traversal cost objectives additively:

$$U_{it}(X) = \alpha U_{ig}(X) + (1 - \alpha) U_{tr}(X). \quad (7.4)$$

We use  $\alpha = .4$ , an educated guess.

| Algorithm       | Runtime (s) | Distance<br>traveled (m) | # View<br>targets | Planning<br>success | Avg update<br>time (s) |
|-----------------|-------------|--------------------------|-------------------|---------------------|------------------------|
| RANDOM-FRONTIER | 6508        | 663.3                    | 2685              | 45%                 | 1.4                    |
| DISTANCE-TO-CAM | 795         | 89.9                     | 353               | 98%                 | 1.4                    |
| INFO-GAIN       | 19379       | 120.5                    | 487               | 90%                 | 25.6                   |
| TRAVERSAL-COST  | 4127        | 89.3                     | 296               | 98%                 | 13.7                   |
| INFO-TRAVERSAL  | 7581        | 61.3                     | 224               | 4%                  | 33.6                   |

Table 7.1: comparison of view selection algorithms on the map completion problem on SIMENV1. Runtime is in many cases the most important statistic. Distance traveled may also be important depending on hardware. Planning failure is almost always due to collisions with obstacles, and occasionally due to failure to find a configuration reaching the initial camera pose for the plan. (In the case of a kinematic-chain robot, this second case is inverse kinematics failure.) “Update time” refers to time spent per run of view selection and motion planning. This includes runs in which there is a current view target and no view selection is necessary; such runs can be very fast. Each successive view target is reached after planning one or more times.

| Algorithm       | Runtime (s) | Distance<br>traveled (m) | # View<br>targets | Planning<br>success | Avg update<br>time (s) |
|-----------------|-------------|--------------------------|-------------------|---------------------|------------------------|
| RANDOM-FRONTIER | 6138        | 638.5                    | 2550              | 38%                 | 1.1                    |
| DISTANCE-TO-CAM | 994         | 118.4                    | 432               | 38%                 | 1.1                    |
| INFO-GAIN       | 42523       | 246.1                    | 975               | 16%                 | 27.6                   |
| TRAVERSAL-COST  | 2807        | 61.3                     | 209               | 64%                 | 13.2                   |
| INFO-TRAVERSAL  | 31852       | 176.9                    | 684               | 1%                  | 46.4                   |

Table 7.2: comparison of view selection algorithms on the map completion problem on SIMENV2. See tbl. 7.1 for explanation of metrics.

Table 7.1 shows numerical measures of the performance of map completion using each of these view selection algorithms. Each algorithm was run once to completion on SIMENV1.

The shortest path length traversed by the camera during the whole run should intuitively be achieved by traversal-cost-based algorithms, and here INFO-TRAVERSAL and TRAVERSAL-COST in fact do achieve the shortest paths. DISTANCE-TO-CAM achieves the smallest runtime to map completion. Multiple algorithms have very high rates of planning success, meaning they successfully avoid selecting next views that are hard to reach from the current one, which would be a negative quality in our continual-replanning framework. The quickest algorithm to run each iteration of view selection and planning is DISTANCE-TO-CAM, which is probably why it completes the map soonest. RANDOM-FRONTIER has a quicker view selection step than DISTANCE-TO-CAM, but has a higher percentage of failed plans, meaning it must replan more times for each view target update.

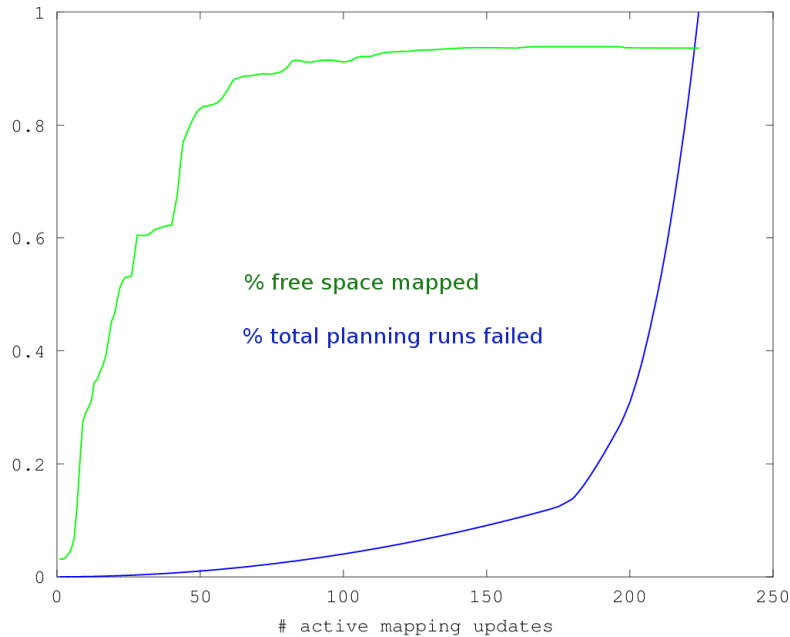


Figure 7.5: map completion (in terms of observed volume) and total planning success rate over time for INFO-TRAVERSAL. As shown in tbl. 7.1, the planning failure rate is 96% at the end of the run; here it appears to be 100% because these numbers include successes as failures. (Each update ends with one planning success.) Qualitative conclusions from this graph are still valid.



Table 7.2 shows numerical results for map completion on SIMENV2, which has qualitatively different topology from SIMENV1 (there is an object resting on two others, creating a hole in the surface). Each algorithm was run once to completion. The runtime numbers are larger than for the less complicated scene SIMENV1, and planning success rates are lower. On this scene TRAVERSAL-COST has the shortest path to completion. It is still the case that DISTANCE-TO-CAM has a shorter update time than any of the other non-random algorithms.

INFO-TRAVERSAL achieves a very short path to view the whole scene, but has a very low planning success rate. Fig. 7.5 shows the completeness of the map produced for SIMENV1 (taken from fig. 7.6 graphed with the (approximate) percentage of failed planning runs after each active mapping update using INFO-TRAVERSAL. Completeness levels off while planning failure becomes more and more frequent over time. One plausible explanation for this behavior is that when most of the volume has been observed, when (we see empirically) there are usually many small unobserved areas left in far-flung regions of the scene, the information gain objective conflicts with the traversal cost objective and faraway areas of the scene containing relatively large unseen regions dominate the top of the list of potential next views. It is possible that the tradeoff parameter  $\alpha$  in the value function would be best changed over time during mapping; this would be an interesting future experiment. Another possibility is that running collision checking before we limit the length of a path has more negative an effect on view selection methods that suggest views anywhere in the scene than it does on those that prefer to suggest views near the current view. To test this hypothesis, we also ran map completion on SIMENV2 using INFO-TRAVERSAL with collision checking run only after path length limiting, to reduce checking of collisions with parts of the map that are not actually traversed until later. This run had a similar planning success rate of 3%, suggesting that overzealous collision checking is not a major cause of planning failures.

We can also look at how quickly each algorithm reduces the size of the remaining free space. All these algorithms eventually view all or almost all surfaces in the map, but they do so at different rates. In fig. 7.6 we show how quickly each algorithm reduces the free space in the scene. We show completion as a function of iteration count rather than of time because each of these algorithms could be tweaked to be faster or slower; it's not at all clear that our implementation of each is the best/most efficient way to implement it, and we want to offset the table above in which we emphasize runtime. For example, both INFO-GAIN

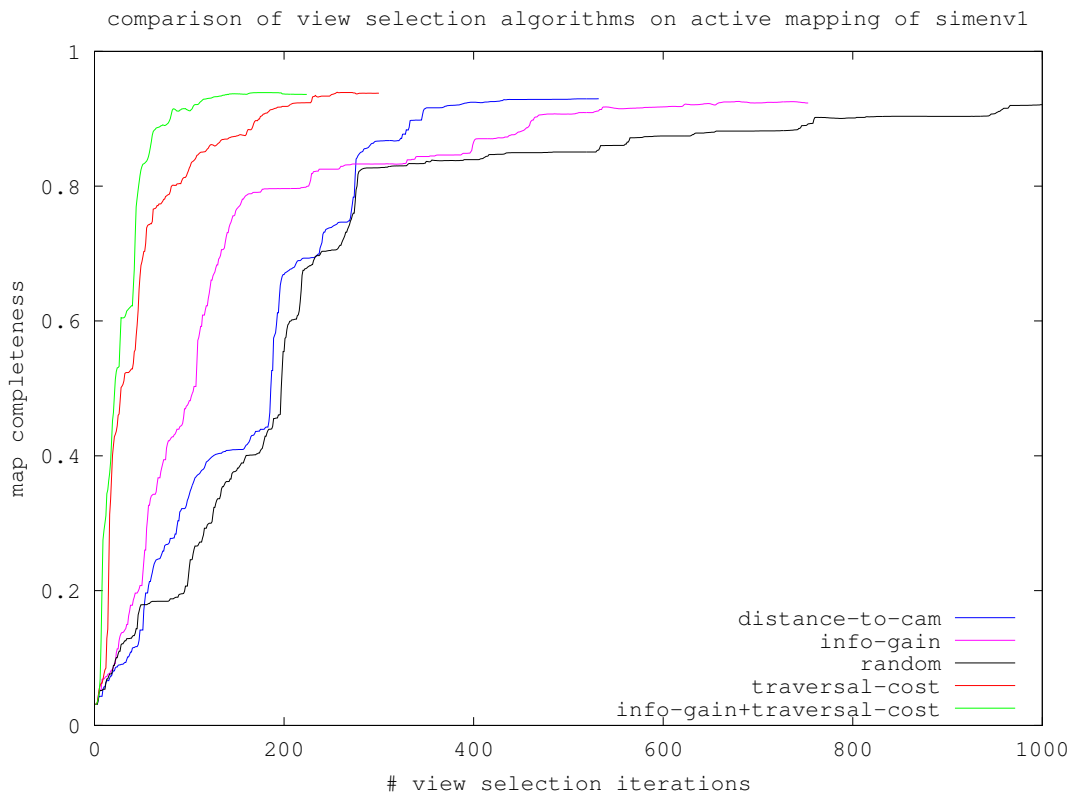
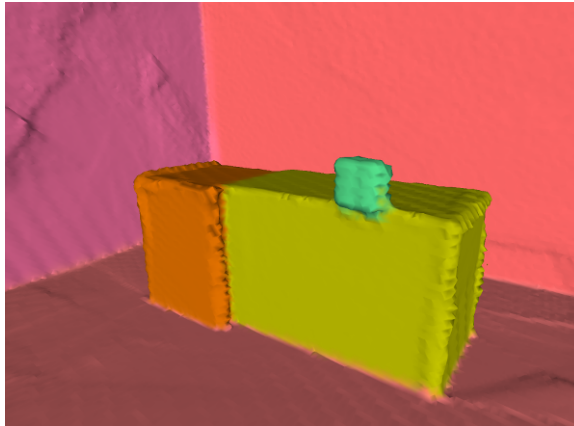


Figure 7.6: completeness of the map as a function of number of active mapping updates executed on SIMENV1. Completeness is measured by free space in the scene that is known free in the map. The x axis is update iterations, which unit is not a linear function of time. The 100% mark (the top of the graph) is, for computational reasons, calculated in a way that slightly overshoots what any algorithm can actually achieve, so although the graph shows each algorithm achieving only about 95% completeness in terms of volume, each algorithm does in fact view all the surfaces in the scene.

and TRAVERSAL-COST could be made faster by greatly limiting the number of options for next viewpoint, as most previous work using those algorithms has. In particular, most of the time in the information-gain-based update is spent in rendering the scene from many viewpoints. However, algorithms like DISTANCE-TO-CAM that use only information local to a mesh triangle to score that triangle, as opposed to spatially dispersed information, allow for considering many view options quickly. This makes the algorithm fit our framework,

which is designed for use with efficient view selection and planning, well. Therefore we choose to build on DISTANCE-TO-CAM as a view selection method for active segmentation purposes.



(a)

Figure 7.7: a map of SIMENV1 made with the DISTANCE-TO-CAM view selection algorithm running on a simulated free-flying camera.

Fig. 7.7 shows a view of a map of SIMENV1 made with DISTANCE-TO-CAM running on a free-flying camera. There are no unobserved surfaces in this map.

#### *Map Completion with a Robot Arm*

| Algorithm       | Runtime (s) | Distance<br>traveled (m) | # View<br>targets | Planning<br>success | Avg update<br>time (s) |
|-----------------|-------------|--------------------------|-------------------|---------------------|------------------------|
| DISTANCE-TO-CAM | 1025        | 89.5                     | 270               | 17%                 | 3.7                    |
| INFO-GAIN       | 22628       | 175.9                    | 562               | 12%                 | 39.4                   |

Table 7.3: comparison of the closest-frontier and information-gain view selection strategies on the map completion problem on SIMENV1 with a simulated kinematic-chain robot. Each view selection algorithm was run once to completion.

For comparison, we run some view selection algorithms on the map completion problem on the SIMENV1 and SIMENV2 environments, as above, but using a kinematic-chain robot

| Algorithm       | Runtime (s) | Distance<br>traveled (m) | # View<br>targets | Planning<br>success | Avg update<br>time (s) |
|-----------------|-------------|--------------------------|-------------------|---------------------|------------------------|
| DISTANCE-TO-CAM | 1002        | 92.6                     | 267               | 25%                 | 3.6                    |
| INFO-GAIN       | 16062       | 138.3                    | 435               | 12%                 | 36.8                   |

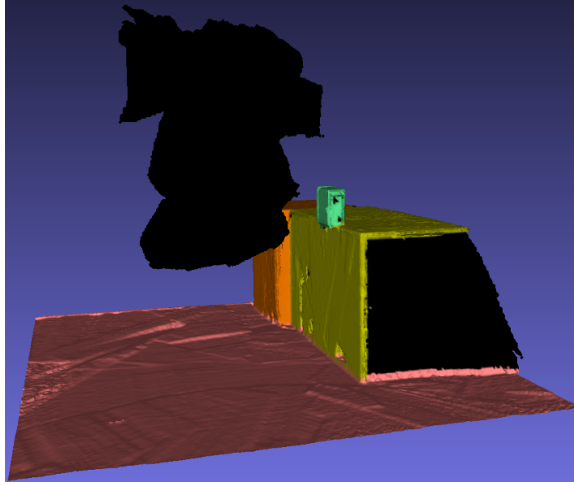
Table 7.4: comparison of the closest-frontier and information-gain view selection strategies on the map completion problem on SIMENV2 with a simulated kinematic-chain robot. Each view selection algorithm was run once to completion.

model. This robot’s configuration consists of the angles of seven rotational joints. We use the OpenRAVE inverse kinematics module, `ikfast`, to implement reachability and collision testing. `ikfast` gives us correct reachability results well over 99% of the time (but not 100%).

We show results for the DISTANCE-TO-CAM and INFO-GAIN view selection algorithms in tbls. 7.3 and 7.4. We selected these two algorithms in order to compare a method using only local to one using global scene information on this robot model. As in the earlier experiment, the local method has better runtime, which is the single statistic we are most interested in in this case, as runtime is a limiting factor due to the mapping speed limitations discussed above.

There are a few differences between these results and those for the free-flying camera model. Runtimes are greater for the kinematic chain, probably because reachability and collision checking are more complicated than for a non-articulated robot. Each algorithm has much lower planning success with the kinematic chain than with the free-flying model; this may be due to the limitations of linear interpolation of joint angles as a planning method. We have used linear interpolation in previous experiments (not described in this document), and it has worked well, but there we have mainly used it for motions through space that is known to be free or assumed to be easily navigable.

Fig. 7.8 shows part of a map of SIMENV1 made with DISTANCE-TO-CAM running on the simulated WAM. There are large areas the camera is unable to see because of robot arm movement limitations.



(a)

Figure 7.8: a cutout of a map of SIMENV1 made by active mapping running on a simulated WAM arm (with the DISTANCE-TO-CAM view selection algorithm). Unseen regions of space are shown in black. The large irregularly shaped unseen area is the region in which the robot is located (its base does not move) and which the camera is unable to see through. The end of the table is also unseen because the arm is unable to move the camera to look at that surface.

### *Object-Centric Map Completion*

Now we consider the problem of object model completion, one way to do active segmentation. In this experiment we wish to model known objects in the scene as fully as possible without manipulating them, and without mapping more of the scene than is necessary. Our view selection framework allows us to design an algorithm for this problem using previously acquired partial information about object boundaries in the scene. To demonstrate the usefulness of using partial segmentation information to inform view selection for the object model completion problem, we compare two view selection algorithms. We again model a camera mounted on a free-flying robot. We first make a map of SIMENV0 using map completion, and use it as the background map against which SIMENV2 was compared via change detection to identify surfaces that have changed. As a baseline, we ran map completion using the DISTANCE-TO-CAM view selection algorithm. To improve results, we propose a change-detection-based view selection algorithm, denoted COMBINED-CHDET-CAMDIST,

which computes view scores for unseen triangles as a combination of distance to camera pose and distance to likely-changed surfaces (defined by having  $p(m) > .6$ , although the exact threshold is not critical):

$$U(X) = e^{-\min(D_m(t(X)), 10) - .4D_p(X)}, \quad (7.5)$$

where  $t(X)$  denotes the triangle that pose  $X$  is computed in order to view, and  $D_m(t)$  gives the geodesic distance on the scene mesh between triangle  $t$  and the nearest likely-changed triangle. (This metric fails when there are unseen regions of space disconnected from the surfaces in the map. This is not a problem for this experiment because disconnected unseen regions are likely to be far from any surfaces, as the camera has to have seen between the two, and here we are concerned with unseen regions close to particular surfaces.) The maximum distance (here 10) is used only to avoid underflow in the result of the exponentiation.

Change detection information is aggregated during mapping in the form of a voxel grid representing the current scene. At each frame processed by mapping, the result of change detection between the current frame and the background map is projected into this grid. We only project information for pixels that have valid depth values in the frame. For each such pixel  $p$ , whose depth in the frame is  $z$  with standard deviation  $\sigma_z$ , we copy per-pixel change detection results to each voxel that projects to  $p$  with a depth  $d \in [z - 2\sigma_z, z + 2\sigma_z]$ . Because we use a voxel grid for this operation rather than a set of frames, as we do in ch. 8, the aggregation operation is constant-time rather than becoming slower as the video being recorded becomes longer.

As a score measure, we use the count of frontier mesh triangles that are within 3 cm geodesic distance of any likely-changed surface in the map being built of the current scene (here SIMENV2). The more frontier triangles are very close to likely-changed triangles, the more interesting surfaces in the scene have not been thoroughly explored yet. The number of such frontier triangles may fluctuate during a run but should eventually be reduced to a very small number (zero, up to map irregularities that occur in real environments). The more quickly this happens, the better the view selection algorithm is at prioritizing completion of partially seen objects. Once the last such triangle is observed, the robot is no longer mapping partially known objects, and until more objects are discovered, mapping proceeds

as in vanilla map completion, or, if the goal of mapping is specifically to complete known objects, mapping can stop.

We ran each algorithm once, as they are both deterministic modulo concurrency effects. The initial camera pose observed parts of all the moved objects in the scene, so that all surface regions of interest initially bordered frontiers. Both algorithms did eventually remove all frontiers near the moved objects. DISTANCE-TO-CAM observed the last frontier triangle near a moved surface at the 3950th mapping iteration, 3788 seconds into execution. The number of frontier triangles near moved surfaces was 213 before the first camera movement, reached a peak of 4074, and did not dip below 213 for the final time until 4351 seconds into execution. COMBINED-CHDET-CAMDIST performed much better. It observed the last frontier triangle near a moved surface at the 796th mapping iteration, 775 seconds into execution. The number of frontier triangles near moved surfaces was 335 before the first camera movement, and stayed under 300 for the rest of the run. (As mentioned earlier, these runtime numbers are larger than those in previous experiments because this experiment required enabling the online change detection functionality in the online mapping system, which slows mapping and thus decreases the speed at which the camera can move.)

### 7.2.7 *Practical Considerations*

We now discuss some practical points in implementing an active mapping system and in using it on a real robot.

Some surfaces (especially transparent surfaces and shiny black surfaces) don't return readings to an infrared depth camera such as we use. The map used during active mapping must include a representation of regions that are unseen due to not returning readings rather than due to not having been viewed yet. We store the number of times each quantized view (quantized view target position and orientation) has been chosen as a target, and avoid selecting targets in a single quantization bin more than  $n = 2$  times.

A camera on a robot arm will point in various directions while it is being moved between targets, due to noncontinuousness of joints in the arm. The mapping system needs to be robust enough to work with long intervals of camera views of ceilings, walls and other relatively untextured surfaces. This may require either disabling mapping during long movements or integration of arm odometry into the mapping system. Patch Volumes currently

only uses visual information. Luckily, unlike in next-best-view planning (e.g. [19]), due to our continuous map aggregation we don't also have the related problem of making sure that each new view sufficiently overlaps existing ones. However, figuring out how to accurately map with general camera motions is still future work.

The simple planners we use in our continual-replanning framework are not guaranteed to be able to produce paths. If we have an almost complete map and there are only a few widely scattered unseen areas left, we may need a sophisticated planner that is deterministically or probabilistically complete, such as an RRT, to guarantee that we can continue to find paths to remaining unseen regions. In general we have a long list of possible view targets at any given time, and we rely on being able to plan to one of the first few targets suggested in order for view selection to be fast, so to integrate a fallback planner with our framework without making it unusably inefficient, we might have to reduce the set of potential view targets. This has not been a major problem in practice so far: most of our experimental runs so far have completed the maps in question despite only having a simple planner available.

In order to communicate effectively with robotic hardware, our framework uses information about the expected runtime of both an iteration of mapping and an iteration of view selection plus planning. In particular, if we expect view selection and planning to last  $T_{plan}$  seconds and each iteration of mapping to last  $T_{map}$  seconds, and we want to avoid moving the camera more than  $D_{map}$  meter-equivalents (see 7.2.6 for a definition of this unit) per mapping iteration, we ensure that no plan (which is intended to be run while the next iteration of view selection and planning happens) is of length more than

$$D_{planmax} = T_{plan} \frac{D_{map}}{T_{map}}. \quad (7.6)$$

At present all these numbers are hardcoded. We estimate  $T_{plan} = 100$  seconds for INFO-GAIN and 1 second for all other algorithms, as INFO-GAIN is so slow that the simulation idles most of the time when a small estimate is used, and we use  $T_{map} = 1$  second from empirical experience. However, any of these quantities could be continually reestimated at runtime. Regardless of the source of this information, it generally won't be perfect, although the more deterministic the planner is, the better our runtime estimate for it will be. It may also be possible to make the system more sophisticated, so that the motion execution module continually checks whether planning is not complete and, if not, sends only a very small part



of the current trajectory to be executed, rather than sending the entire output of planning at once.

### 7.2.8 *Summary and Future Work*

In this section we have presented a framework for continual view selection and motion planning in an online active mapping framework, in which the robot can move the camera. We assign a value to each scene element that represents a frontier area in the 3-D map, and prioritize planning to scenes with high value. We use a simple and fast planner in order to keep the camera in motion as much of the time as possible. We use this framework for the map completion problem as well as for a variant emphasizing completing models of previously partly seen objects, which is useful for active object segmentation.

Our planning so far has encompassed kinematic-chain robots, such as robot arms, and free-flying robots (an approximation to autonomous helicopters), but has not included robots with both wheels and arms. Planning for these is very computationally inefficient [86]. It may be that an online-updated probabilistic roadmap is the most useful avenue of attack for bringing motion planning for such a robot into our framework, which assumes that basic actions such as planning are relatively efficient and can be run often.

The operation of removing occupied volume from the background map that is used in ch. 6 might be better done volumetrically, as opposed to the current rendering-based approach, which throws away some information about free space during the surface removal operation. We have just introduced another operation that is best done in a volumetric map: aggregating change detection results over time, which we use to help define some value functions, is more efficiently done in a fixed-size voxel grid than in a set of images. Each image is much smaller than the voxel grids we use for mapping, but that representation increases in size as new frames are processed. For now we implement change detection aggregation in our own volumetric representation on the CPU, due to Patch Volumes being closed-source, but ideally both these operations, and probably others, would be implemented as part of a GPU volumetric mapping API.

### 7.3 *Interactive Segmentation: Causing Object Motion*

For purposes of mobile robotics, it’s useful to have the notion of what it means to be an object depend on the capabilities of the robot in question. In this way we make it possible for hypotheses about the shape, identity or other properties of individual objects to be produced *and verified* by a robot. Our robot consists of an RGB-D camera and a manipulator arm with a three-fingered hand, is generally able to (1) see, (2) touch and (3) move or pick up objects, so for our purposes an “object” is a movable object light enough to be moved by the arm and that is not invisible to our camera or depth camera. (Transparent objects, for example, are rarely seen by infrared depth cameras.)

The ability to move the camera to get a more complete view of the scene does in general improve our ability to see all sides of an object, but it fails to see all sides of all objects in heavily occluded scenes. In general, to see all objects manipulation of objects is necessary. Therefore we now discuss preliminary work on object segmentation using manipulation actions. We choose the push as the type of manipulation to use, due to its simplicity and robustness to the response of objects to being touched. (Picking-up actions fail much more of the time than pushing actions do.) The novelty of the approach we outline here is to learn where to push on the fly by performing online learning of models for manipulation-related attributes at locations on surfaces in the scene. This is intended to generalize knowledge gained through experimentation in one scene to other scenes with dissimilar visual content, such that methods such as point feature matching that work for motion estimation in some scenes may no longer work. Previous work on online learning of low-level visual models for mobile robots has, to our knowledge, been limited to learning what local patch appearances symbolize obstacles [138], which is not very generalizable across scenes.

#### 7.3.1 *Related Work*

Many authors have worked on interactive object segmentation using both short- and long-time-duration motion estimation techniques. Kenney et al. [99] push objects in a plane, with a fixed overhead camera, and using a sideways arm motion to keep the arm from obstructing the camera’s view of the objects. The arm motion is prespecified. Point feature matching is used for motion estimation. Fitzpatrick [57] uses change detection on images taken before and after a push rather than feature matching. Katz et al. [96] push objects in

3-D and track 2-D point features. They use various static cues and features of the points' 2-D motion over time to cluster sparse point features into multiple rigid bodies. The camera is fixed and the arm motion is prespecified. They extend this algorithm to multiple pushes by asking a human to decide what manipulations to make [97]. Bergstrom et al. [13] also fix automatic object segmentations by interaction with a human. Ude et al. [176] choose groups of point features as hypothesized objects and push these to verify. They push in a random horizontal direction, and they do not present a termination criterion. Chang et al. [32] push a pile of multiple objects to segment it into individual objects. They select pushes in the horizontal plane. Importantly, they use a learned function over feature matches to decide when to stop pushing. All these approaches but [57] assume visual point feature matches are available.

There is also recent work that does not rely on point feature matching to provide correspondence over time. Hermans et al. [80] attempt to separate all objects in a scene from each other using pushes. They model the directions in which each spatially connected cluster of points in an RGB-D scene has been pushed by a robot, using the major axis of the 2-D shape of a point cluster to determine the cluster's rotation, and resetting clusters formed when multiple objects close in space become separated after a push. Van Hoof et al. [178] segment a pile into individual objects using pushes with only approximate temporal correspondence, using single-color segments (on the order of dozens per scene) as primitives. They accumulate evidence for two segments being part of the same object when the segments move during the same push. They use rough color-based motion estimation for segment correspondence.

### 7.3.2 Algorithm Overview

Suppose a robot is introduced into a novel scene. After taking sensor readings of the current scene configuration, to gain further information it must manipulate the scene. We push surfaces in an attempt to find movable objects by making them move. If we successfully move an object, we can locate it in views of the scene before and after the push; here we do this using scene differencing.

Once we have a moved object localized, we can use it as training data for online learning of mapping-related models. In particular, we maintain models of the *pushability* and the *objectness* of surfaces in the scene. Objectness tells us what surface points are likely to be

near the centers of objects, and so be good candidates for pushing if the goal is to cause objects to move. Pushability tells us how likely a local surface patch is to be a useful push point, and allows us to avoid repeatedly pushing immovable scene elements. These models are used in selecting future manipulations.

Our proposed life-long object mapping algorithm consists of iteratedly planning a manipulation, trying to execute it, and updating our models of push-related attributes given what we can infer about the motion we have caused.

### 7.3.3 *Manipulation by Pushing*

The manipulation primitive we use is a push: the robot chooses a point on a scene surface (the *push point*) and a direction in which to try to move it, extends a finger, and moves the finger in the chosen direction for a predetermined distance or until it encounters resistance. Our manipulator is not equipped with a force sensor, so during the part of the manipulator’s motion in which we expect to encounter resistance, we repeatedly move a short distance and test how different the manipulator’s measured and expected positions are. If they become too far apart, we stop the motion in order to avoid damaging the manipulator. If we have to abort a motion, we don’t assume that no object movement occurred.

We find that the direction and length of the planned push are not a very good approximation of the actual object motion, particularly because pushing a heavy object may cause our robot (an arm on a mobile platform) to move backward slightly, so we need a way to segment the pushed object given snapshots of the scene before and after the push. (We choose this framework rather than that of tracking moving surfaces during the push because this is a simpler first step.) After we detect motion, we focus on the area surrounding the surfaces that moved, repeatedly pushing near them until we think we have identified a single object as having moved. Once we believe we have segmented a single object, we add it to our store of training data, then repeat the process. We terminate once all the surfaces in the scene that we believe (as defined by the pushability model) to be movable have been moved.

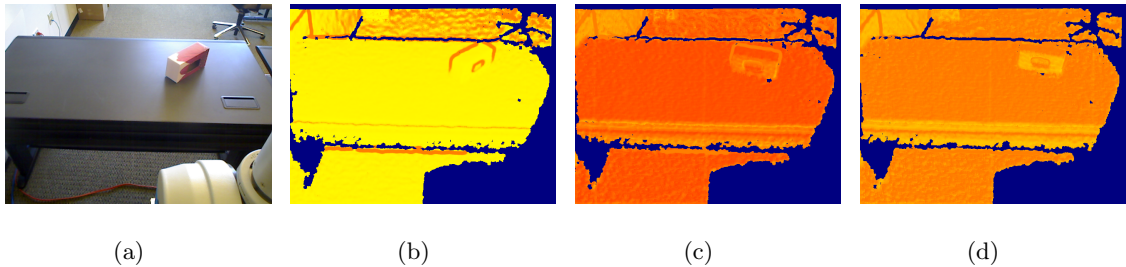


Figure 7.9: the progression of the learned surface pushability model over a series of pushes of the same object. (a) the scene before pushing. The other three images show the model as a function of surface point, shown for the scene *before* each push. Red  $\leftrightarrow$  yellow = low  $\leftrightarrow$  high pushability. (b) the learned model after one push. The robot pushed the box from the side, and now thinks most surfaces are pushable. (c) the learned model after a second push attempt. The robot pushed down on the top of the box and did not cause motion. It now thinks horizontal surfaces are not pushable. (d) the learned model after another successful push, of the side of the box. Pushability values are moving toward 0.5 and vertical surfaces are considered more pushable than horizontal ones.

### 7.3.4 Object Modeling for Manipulation

To inform planning of future manipulations, we continually update learned models of characteristics of the scene as functions of 3-D location (only defined for locations on surfaces). These models, in addition to modeling the current scene, could be used to initialize models for future scenes (for example, the living room of an apartment whose kitchen we are currently modeling). After each manipulation attempt, we retrain our models and update our estimates of the corresponding characteristics at all surface points in the current scene (which may have changed as occluded surfaces come into view during motion). We acquire few training data from each manipulation, so we expect that our models will perform suboptimally for the first several manipulations in a scene.

The *objectness* model suggests likely object centers. We learn the probability that a local surface patch is on the boundary between an object and the (non-manipulable) background. We create a list of plausible object centers in the current RGB-D frame using approximate boundary orientations of points identified as likely boundary points. The other model tells us the *pushability*, in  $[0, 1]$ , of each point’s local neighborhood. After each push attempt we know how far we succeeded in moving the manipulator. We threshold this distance to decide

whether the push was “successful”, and create a training example for the neighborhood of the push point.

Fig. 7.9 shows the progress of pushability model learning from the first few pushes of an object.

### *Selecting Manipulations*

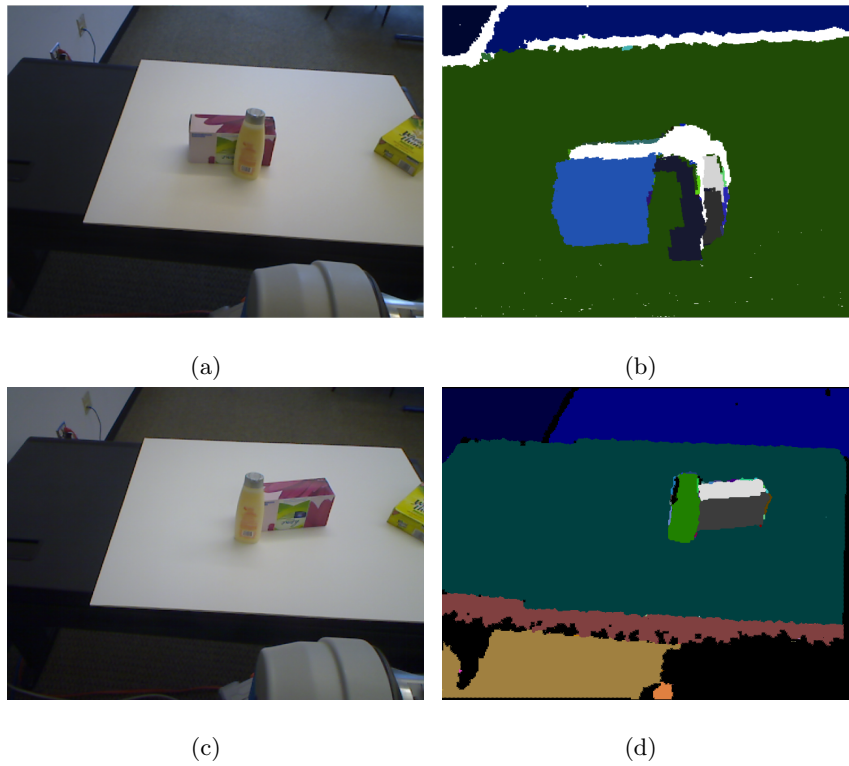


Figure 7.10: an illustration of using the gPb hierarchical segmentation system to help suggest push points on objects. (a), (c) images of two scenes the robot is presented with. (b), (d) the segmentations produced by our customized gPb implementation for these two RGB-D scenes. In (d), the large light and dark gray segments (shown with arrows) have been chosen to have their connecting boundary pushed by the robot.

Intelligently choosing what surface in the scene to push and how to push it is necessary for active segmentation to be a usable tool for scene understanding. If a robot continually tries to explore parts of the scene that won’t give it information, it can waste arbitrary

amounts of time and power. We use our learned models of scene characteristics to decide where and how to push: the objectness model gives us surface regions likely to be on objects and in areas useful to push (near the middle of an object). Some of these regions will in fact be on the background, so we weight the objectness of each point with its pushability to choose the push points most likely to give us useful information (since pushing immovable surfaces yields very little information).

We have also been experimenting with a manipulation selection approach based on generating a small set of reasonable hypotheses and choosing the most likely of them to verify. Hypothesis verification is a common strategy in mobile robotics and has been shown to be theoretically useful for creating efficient algorithms [91]. In particular, we have tried customizing a hierarchical segmentation generator (the global probability of boundary approach [7], developed for color images) to produce static segmentations of RGB-D scenes; the idea is then to select edges between adjacent segments somewhere in this tree on which to push. Fig. 7.10 illustrates this idea.

### 7.3.5 *Directions for Future Work*

In this section we have discussed maintaining information about the current scene and online learning of models specific enough to help understand the current and future scenes the robot will encounter. Our high-level goal is to be able to understand a novel scene in terms of known and novel objects as quickly as possible. Therefore it will behoove us to keep a library of known objects to match newly observed partial models to. Manipulation will be required both to model new objects and to clear clutter to improve the accuracy of processing occluded known objects.

One goal we have is to track objects over time, so we want to be able to register pre- and post-push object views (3-D segments). Another reason to have registration is to ease finding the object after motion, as done in sec. 5.3. Also, currently once an object has been pushed and segmented, we are unable to remember that we have already processed that object, and if it moves later due to pushing of other objects, we may eventually start pushing it again.

## 7.4 *Summary*

One of the major benefits of using mobile robots for vision is that a robot can choose how to explore its environment, which a passive vision algorithm cannot. In this chapter we have discussed two ways in which we can make use of having a robot available to improve our visual understanding of a scene. Firstly, controlling the movement (or, in general, hardware settings such as zoom or software settings such as exposure) of the camera (or other sensors) greatly increases the number of scenes we can fully model. Secondly, by manipulating objects directly we can obtain views not physically available in the original scene. We make sure that our hypotheses about individual objects are such that the robot is capable of verifying them through manipulation. We have shown that we can make use of the ability to move the camera to complete a map or to prioritize looking at certain parts of the map (e.g. known objects). Our work on interactive segmentation, which focuses on making manipulation selection more intelligent than it has been in past work, is less mature and it is not yet clear how much we can improve the accuracy of segmentation by using manipulation.



## Chapter 8

## OBJECT DISCOVERY

**8.1 Introduction**

The methods of ch. 4 allow us to segment objects in many scenes, but do no or little data association: in the case of change detection, we have no idea where each object moves to from one time point to another, and in the case of the feature matching and flow-based algorithms, we have only data association between two times. One way to extend feature matching or flow-based segmentation over time to get *object persistence* would be to chain correspondences over contiguous pairs of time points. However, we can also take a more general approach to object persistence that allows us to link object views acquired with any segmentation method. This is the approach taken in this chapter.

There are many reasons for which we want a notion of object persistence. We can get an idea of the number of unique objects we’ve seen in a particular location. We can estimate distributions of the location of a particular object over space: how often is object #14 seen at location #3? Or we can, given an object ID, say where it was first or last seen.

We achieve object persistence through *object discovery*. Given two or more mapped scenes with small changes between them caused by objects moving, we can extend the two-scene segmentation of sec. 4.3.3 to a multi-scene approach whose segmentation improves as more scenes are available. Object discovery is the step of deciding which of these views on objects represent the same object. We use a graph-based (batch) clustering method over segments, so we need a measure of similarity between two segments. For this purpose we test a variety of 2-D and 3-D segment matching techniques using a change-detection-based error as well as shape and visual descriptors. Due to our non-feature-matching-based approach, we can robustly detect objects and their motion between scenes even when objects are textureless or have non-distinctive shapes.

In this discussion, we use change detection over previously built maps to provide object views to be clustered. However, the source of segments is orthogonal to the clustering stage, so any method we have discussed in a previous chapter could be used to provide

input to object discovery. Another design choice we have made in this chapter is to use a batch clustering method, which disallows us from incrementally updating the clustering after seeing a new segment except by rerunning clustering of all previously known segments. There are at least two ways in which we could change this in future work: use an online clustering algorithm, such as one of those discussed in [47], to add one segment to the clustering at a time, or make some number of hard decisions about which segments are the same object after each clustering run, while leaving some other decisions to be decided after later runs.

After discussing related work, we develop the steps of our algorithm in section 8.3. Experimental results are provided in section 8.4, followed by conclusions.

## 8.2 *Related Work*

Previous work on representing the static and dynamic parts of maps separately has been restricted to 2-D, whether using color or depth sensors, and has generally avoided the problem of segmentation. Biswas et al. [18], working with planar laser scans, assume a known number of objects and segment objects by connected components of occupied grid cells. Haehnel et al. [71] run mapping with 2-D laser scans, then approximate each 3-D object as the set of 3-D measurements within a 2-D bounding box. Konolige et al. [105] represent a map as a set of views (images) from a color camera, and cluster views by location. Using their methods, it is possible to represent objects indirectly by partitioning each location’s set of views according to similarity, but they don’t explicitly model objects or segment images, as object-level representation is not one of their goals.

Usually computer vision work using objects acts at one of two levels: instance and category. Object instances are typically modeled with point features and their geometric consistency (enforced with RANSAC [56]), although more complex methods using pairwise constraints have been proposed (e.g. [52]). Another approach is cosegmentation, which jointly finds regions in multiple images corresponding to the same object. Joulin et al. [93] cosegment by maximizing the margin between object and non-object pixels over a set of images. They have particular trouble when images have similar backgrounds. We avoid this issue by using motion for segmentation; we *require* that different scenes have the same background.

By contrast, there has been a wealth of work on object *category* discovery. Grauman and Darrell [68] assign each image a category by performing spectral clustering with a spatially aware kernel over sets of local features. Russell et al. [152] extract a “soup of segments”, model each as a histogram of visual words, and perform topic discovery on the set of segments. They can localize multiple instances per image. Graph mining techniques have also been used: for example, Kim et al. [100] create a graph among local features, with edges specified by feature matching.

Working with depth data, Ruhnke et al. [151] align object views by using correspondences between local depth-map features, without color information. Their keypoints are corners in the depth map; they assume the shape of the edges of the 2.5-D object views is enough to distinguish both the object identity and the location of each point on the object. They too ignore the problem of segmentation, using connected components on point clouds to produce candidate objects. The objects they disambiguate are not very similar. Shin et al. [161] instead do bottom-up segmentation in a single scene. They discover objects using depth but not color, using ICP initialized by matching spin images. They do not demonstrate discovery of small-scale objects such as our dense scene reconstruction allows us to find. In [175] they extend their method to use co-occurrences of matching segments to discover multi-segment objects. Mason et al. [128] use change detection on visual point features, finding spatially localized groups of features that disappear between viewings of the same scene, to define sparse spatial extents for objects. Unlike us, they segment different frames of the same video sequence separately, relying on the clustering step of object discovery to decide that segments from temporally adjacent video frames are the same object.

There has also been work on instance discovery from video. Liu and Chen [119] combine visual topic modeling with a motion model for the foreground segment to track a small moving object. Southey and Little [164] find multiple objects and don’t assume they appear in all frames. They get sparse depth points from passive stereo and approximate motion from sparse optical flow. Rigid motions are fit to each frame pair using local features. They oversegment each frame and have segments vote for motions. In both cases, motion within a single video is used for segmentation; we use motion between multiple videos for segmentation and use the frames of each video to provide spatial consistency.

At a high level, the algorithms of [151] and this chapter are both generate-and-test algorithms of the sort also used for 3-D alignment (e.g. [3]), and so consist of the same four steps

as outlined in table 8.1. We use our scene differencing operation as much as possible, and avoid assuming there are enough distinctive point features for feature matching. Additional differences are our use of color information, our multiscene optimization for segmentation, and our fuller experiments.

| Step                     | Ruhnke et al. detail                      | This chapter detail         |
|--------------------------|-------------------------------------------|-----------------------------|
| 1. Segmentation          | connected components                      | scene differencing          |
| 2. Hypothesis generation | create 2-D views;<br>2-D feature matching | ICP with<br>random restarts |
| 3. Hypothesis scoring    | ICP error                                 | scene differencing          |
| 4. Clustering            | repeated Olson [137]                      | n-cuts                      |

Table 8.1: High-level structure of this chapter and [151].

### 8.3 Multi-Scene Analysis

In sec. 8.3.1, we describe how to reconstruct 3-D scenes from video. In sec. 8.3.2, we show how to use motion cues to find possible objects. We calculate pairwise similarities, and use them for clustering 3-D segments, in sec. 8.3.3.

#### 8.3.1 3-D Reconstruction of Single Scenes

Given a set of scene visits in the form of RGB-D video sequences, we run a modification of RGB-D Mapping [75] on each visit to produce the set of camera poses corresponding to all video frames. RGB-D Mapping consists of 1) visual odometry using SIFT features assigned to 3-D points, 2) loop closure detection, also using SIFT with 3-D locations, 3) pose-graph optimization over contiguous-frame and loop-closure edges, and 4) bundle adjustment over the whole map. The RANSAC variant we use in steps 1 and 2 optimizes reprojection error rather than 3-D error as in [75]. Another change we have made to the algorithm is to run bundle adjustment over each contiguous-frame pair following RANSAC in the visual odometry step.

As in [75], we represent surfaces using *surfels*, small planar patches with color and orientation information. We make use of uncertainty in sensor readings by fitting Gaussian

models for the position, color and orientation at each surfel, as we find that this improves the performance of scene differencing. We globally align all scenes using visual point feature matching followed by ICP, as in [76].

### 8.3.2 Map Segmentation

In a previous section (sec. 4.3), we have introduced an algorithm to use RGB-D sensor readings to probabilistically decide which surfels in a scene are unchanged in a second scene. We run inference in a Markov random field (MRF) over these probabilistic judgments to create a binary (moved vs. not moved) segmentation of each scene. We use this pairwise scene differencing algorithm to segment each visit to a scene by using change detection with respect to each other scene in the dataset. We then use the multi-scene segmentation approach of sec. 4.4 to make use of as many maps of the scene as we have to segment each map. We showed in sec. 4.4 that we can get noticeably nicer segmentations from many visits than from two.

### 8.3.3 Object Discovery

In the computer vision community, object discovery is the process of partitioning a set of views (in our case a view is a segment extracted from a single scene) such that those in the same group represent the same object. Many object discovery algorithms use a clustering step that relies on a similarity measure between pairs of views. One unique aspect of our work is that we have access to both 2-D appearance (one RGB-D view in each video frame) and 3-D shape (reconstructed by merging a video sequence). We can thus match observations (segments from the multi-scene MRF) using both 2-D view-based matching and 3-D alignment-based matching.

### 2-D Matching

Fig. 8.1 shows examples of 2-D *views* extracted from the segmented 3-D scenes. To compare 3-D segments using these views, we extract features over entire segments, rather than matching point features calculated at interest points (such as in [39]). We have two reasons for this: the image regions corresponding to objects are generally small and have few features and even fewer reliable ones (due to region-edge effects), and some of our objects are

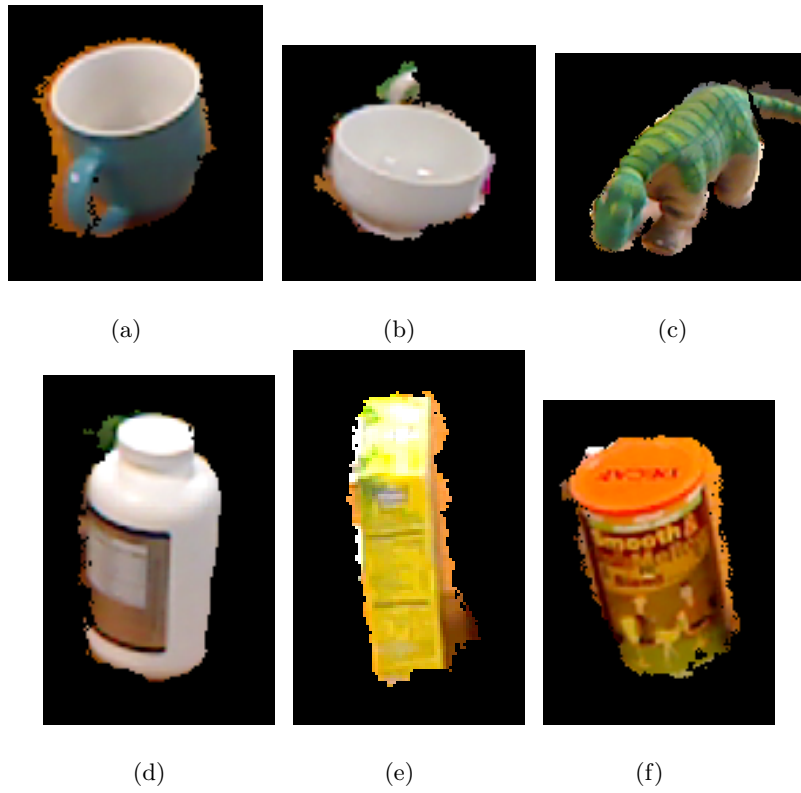


Figure 8.1: examples of video frames masked by the projections of 3-D segments into camera coordinates. These are not entire frames, but cutouts of width about 100 pixels. We compute 2-D features over image regions like these.

textureless and have no reliable point features.

We use efficient match kernel (EMK) features [21] calculated over sets of kernel descriptors [20] to represent our segments. Kernel descriptors are a generalization of histogram-like features such as SIFT and achieve leading accuracies on image category recognition [20]. We have both color and depth information available, so we calculate gradient kernel descriptors over both color and depth, and color kernel descriptors over color. (These three descriptors had the highest learned importance in our preliminary experiments.) We calculate features over regions defined by projecting 3-D segments into each camera’s frame rather than over the whole frame. For EMK we use a three-level pyramid (1, 2 and 4 bins in each of  $x$  and  $y$ , respectively) and five hundred visual words. We get our visual words by running  $k$ -means on a large set of kernel descriptors extracted from some of the scenes in one of our datasets, but have found that the number of visual words created (we use 500) is more important than their source. The output of this step is a 10,500-D vector for each kernel descriptor type for each 2-D segment ( $1 + 2 \times 2 + 4 \times 4 = 21$  spatial pyramid cells, with a 500-D vector for each cell since we have 500 visual words; concatenate these).

We want descriptors for an arbitrarily shaped image segment rather than an entire image, so we project the 3-D segment into each 2-D image in its scene to select a set of pixels, including a 1-pixel boundary around the resulting masks to capture gradient information at the edges of the object. EMK requires a rectangular image window; we set non-selected pixels in the window to black, for lack of an obvious “correct” way to handle this issue.

We run logistic regression for each of  $S_1$ ’s frames to determine the closest frame in  $S_2$ . We use the following features: distances between EMK vectors for each feature type; a boolean feature that is true iff the selected match is ‘outlier’ (a label used when no frame in the second sequence matches well); a boolean feature that is true iff either the selecting frame or the selected match frame does not contain the segment; and a boolean bias feature that is true unless the outlier feature or the invalid feature is set. We find that the  $L_1$  distance between EMK vectors works better than the  $L_2$ . We set the weights by hand (we find training makes little difference). The best way we have found to combine these frame-to-frame match scores into a single score for the segment pair is to take the 90th-percentile score. This is essentially the best score over all frame pairs, but allowing for a few high-scoring matches due to chance.

### 3-D Matching

We also consider matching the 3-D segments we get from the segmentation step. We run an RGB-D variant of iterated closest points (ICP)  $n$  times for each pair of point clouds, initializing by setting the centroids equal and choosing a rotation uniformly at random. For speed we use point-to-point error [15] rather than the point-to-plane error of [33]. Color information is used by requiring each point to correspond to a point whose color is in the same hue-saturation bin (we use 4 bins in hue, 4 in saturation). We can score the result of each ICP run by taking the average 3-D error over points if at least half the points have correspondences, or a very large number if not. (If ICP is unable to match a majority of points, its error will not be a reliable measure of similarity.) Another error measure, designed to be more accurate, is an adaptation of the output of scene differencing [76]. We compute the probability of change at each point  $x$  in  $S_1$  and associate each  $x \in S_1$  with a point  $c(x) \in S_2$  as in ICP, such that some points have no correspondences. The score  $J(X)$  for the alignment  $X$  is then the sum of the pointwise change probabilities weighted depending on correspondence:

$$J(X) = \alpha \sum_{x|c(x) \in S_2} p(x) + \sum_{x|c(x) = \perp} p(x),$$

where  $\alpha = 10$  works well. Using the scene differencing score significantly improves the reliability of the ICP matching.

Matching via 3-D alignment has the advantage of also providing a hypothesized alignment between matching segments. We could estimate an alignment from our 2-D matching by supposing that the object orientation is similar throughout the two videos and averaging, or running RANSAC over, transformations estimated from each individual frame match, but we find that the resulting alignments are bad enough that we need to run ICP afterward to make them useful. Some example transformations, good and bad, that we get from the ICP-based matching are shown in fig. 8.2.



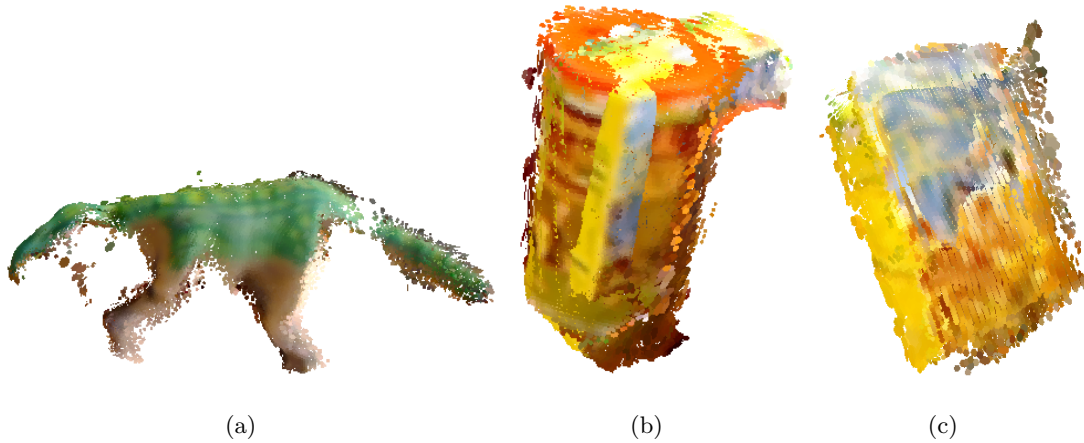


Figure 8.2: example ICP alignments. Each is the best of a large number of ICP runs with random initialization. (a) The two dino clouds line up well. (b) The ICP error for these clouds is very low. The segment of the yellow box includes an orange section of this orange can, which appeared next to the box in the scene the yellow segment was segmented from. (c) The ICP error for these two segments, which are the same object, is very high because one of the clouds consists of only one side of the box and the other has three sides.

### *Spectral Clustering*

The pairwise similarities computed from 2-D and 3-D matching can be very noisy and inconsistent, partly due to low-resolution data, partial occlusions of objects, and errors in differencing and segmentation. Clustering is needed to enforce transitivity in the same-object relationship between segments. We use *multiclass spectral clustering* [196], which minimizes the K-class Normalized Cuts criterion

$$ncuts_K(\{V_k\}) = \frac{1}{K} \sum_{t=1}^K \frac{\text{link}(V_t, V \setminus V_t)}{\text{degree}(V_t)},$$

where the total set  $V$  is partitioned into  $K$  subsets  $\{V_k\}$ , link is the sum of edge weights between two sets, and degree is the set of edge weights within a set. A discrete solution to the K-class Normalized Cuts can be found efficiently by finding rotations in the eigenvector space using singular value decomposition [196]. We find that spectral clustering overcomes many sources of noise in the pairwise links and discovers challenging objects with large occlusions and without distinctive texture or shape (see examples in Figure 8.6).

We have tested three variants of spectral clustering: kernel k-means [48] (our implementation), self-tuning spectral clustering [197] (authors’ implementation), and multiclass normalized cuts [196] (authors’ implementation). Normalized cuts reliably outperforms the other two (except for self-tuning spectral clustering in very-low-recall regimes), so we use only it in our evaluation (and provide some comparative results there).

## 8.4 Evaluation

Our RGB-D camera provides registered color and depth images at high frame rates; for this paper we use 10 Hz. Each scene was collected separately (the equivalent of assuming visits have been pre-segmented from a continuous video) to reduce storage. The camera was carried by a human. We test our algorithm on two datasets, one (A) containing 10 scenes and one (B) containing 9. The two have different sets of moved objects: B has more textureless objects than A, and more objects overall (15 versus 8). The average scene length is 140 frames for A and 230 frames for B, at 10 Hz. To show the power of our approach, we included similarly colored and shaped objects. For example, B contains three mugs and four bowls, none with much texture and all with white/blue color schemes.

### 8.4.1 Segmentation

To evaluate our segmentation algorithm, we manually labeled objects in each scene and marked which objects moved between each pair of scenes. We labeled only dataset A due to time constraints. We compare the two-scene “pairwise” MRF of [76] to our “multiscene” technique. For space reasons table 8.2 includes only aggregate statistics over surfels in the 90 scene pairs in dataset A. Most surfels in any scene pair do not move; the “always-not-moved” classifier attains over 97% accuracy. As can be seen, multiscene reduces the error by over 25% and increases both precision and recall (calculated w.r.t. surfels that truly moved) noticeably. (Over our dataset, this change in precision translates to 320,000 false positives avoided.) An example of the improvement we see from the multiscene MRF is given in Fig. 4.9; five other scene pairs in dataset A show similar improvement, and for no scene pair is multiscene numerically or visually significantly worse than pairwise.

We also demonstrate the scalability of our multi-scene formulation. The graph-cut inference we use does not require that potentials be stored in memory, which means that

| algorithm  | # surfels | accuracy | precision | recall | FP    | FN    |
|------------|-----------|----------|-----------|--------|-------|-------|
| pairwise   | 38.7M     | 0.991    | 0.971     | 0.901  | 1.13M | 3.84M |
| multiscene | 38.7M     | 0.993    | 0.979     | 0.927  | 0.81M | 2.83M |

Table 8.2: performance statistics for pairwise and multiscene differencing MRFs over all scene pairs in a dataset. FP and FN are false-positive and false-negative counts. We reduce FP by 28% and FN by 26%.

inference can run in a small amount of memory even with exponentially many labels, and that the time to set up inference varies very little as a function of the number of labels. Fig. 8.3 shows the decrease in label set size we get via the method discussed in sec. 8.3.2 as well as the resulting inference runtime for scene 1 of dataset A. Inference setup time is about 21 seconds (not included in the plot).

After the segmentation step, we manually removed segments that were not whole objects, such as table legs and pieces of floor, as well as segments containing (parts of) multiple objects, in order to get better differentiation among methods on relatively clean data. This left us with 48 and 64 segments for datasets A and B, respectively, out of 116 and 183 segments total.

In addition to the surfel maps used here, in sec. 6.5 we test this segmentation method on volumetric maps built with Patch Volumes. The method finds all objects that move in that dataset. Many of the false positives seen there are due to imperfect construction of graphs over surface elements: relatively small disconnected pieces of surface take on non-background labels relatively often, which they would not if the graph of the scene were a single connected component. We have not thoroughly experimented with methods of connecting the entire graph.

#### 8.4.2 Pairwise Segment Matching

Although our goal is clustering segments, we can also evaluate the intermediate step of pairwise segment matching. Each of our matching methods gives us a score for each pair of segments. We evaluate by taking scores over a threshold to specify true matches. We can thus calculate precision and recall of a score over the set of all (ordered) segment pairs

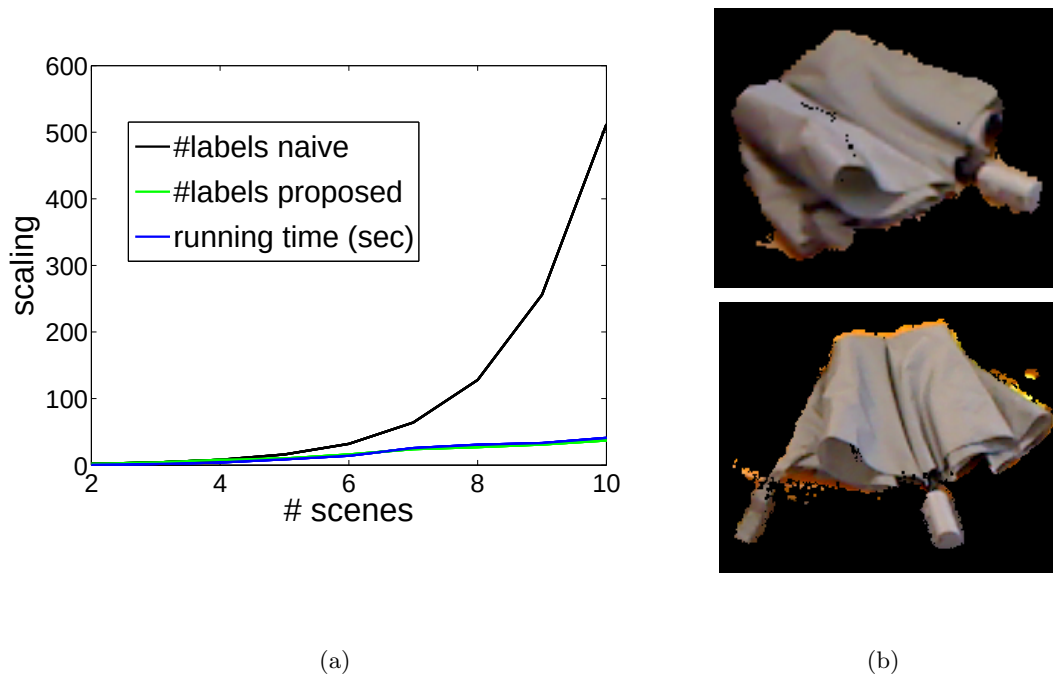


Figure 8.3: (a) scalability of multiscene differencing with number of scenes. The black curve gives the number of labels used by the naive algorithm. The green curve is the number of labels we use. The blue curve (right on top of the green) gives runtime in seconds. All have the same y-axis scale. (b) An example of a segment pair that 2-D features have trouble matching. The gradient distributions of the two 2-D segments are very different because the views are from different orientations. The object is an umbrella. These are crops of widths 140 and 228 pixels.

in a dataset for each of a range of thresholds, and we do so in Fig. 8.4. Since the 2-D and 3-D alignment methods described in sec. 8.3.3 use very different sources of information, we expect the combination of their scores to perform better than either alone. As Fig. 8.4 shows, this is the case for both of our datasets when we use ICP-based scoring as the 3-D component. (To get a combined score for a segment pair, we normalize both score distributions by mean and standard deviation, then take a weighted average. For Fig. 8.4 we weight the ICP score by 30.) Combining the differencing-based score with the EMK score does not improve on differencing-only scoring, so we don't show it.

When we developed the differencing model we hoped to build the whole object discovery pipeline on it: segmentation, registration and matching. The fact that differencing-based

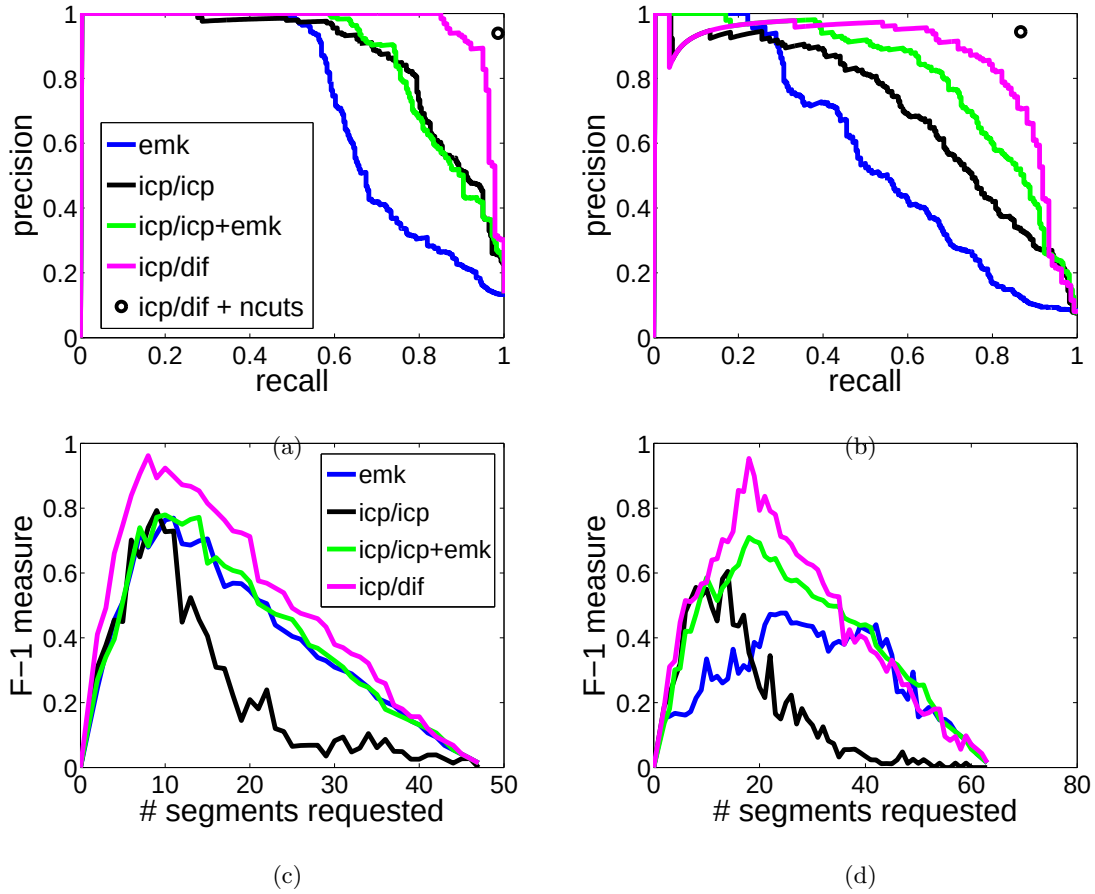


Figure 8.4: (a), (b) precision-recall curves for pairwise matching on datasets A and B respectively. Blue: 2-D visual-feature-based matching; black: ICP with ICP-error scoring; green: combined visual features and ICP; magenta: ICP with differencing-based scoring; black circle: best result of spectral clustering on differencing-based scores. (c), (d) performance of spectral clustering on the two datasets (same color scheme). The black circles in (a), (b) come from the magenta curves here. F-measure is the harmonic mean of precision and recall. True numbers of clusters are 8 and 15.

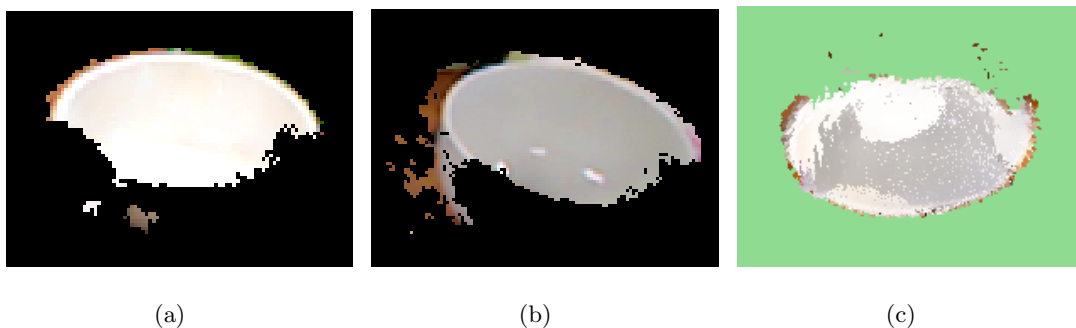


Figure 8.5: segments from different objects (two white bowls) that differencing thinks very likely to be the same object. (a), (b) images of the bowl segments; (c) the point cloud alignment of the two segments (upside down w.r.t. the other images).

transform scoring performs so well suggests that we can further improve matching performance by incorporating differencing in the registration step. We would like to optimize a differencing-based objective rather than the ICP error measure, and in a way that correctly deals with occlusion, rather than in a suboptimal way such as simply changing the objective in the optimization stage of each ICP iteration.

2-D and 3-D scoring methods make many of the same mistakes: false negatives happen for segment pairs with very different colors or that cover different parts of an object, and false positives happen for textureless objects of similar shape and for textured objects of similar color. Visual features make mistakes that ICP doesn’t on segments containing different parts of an object—e.g. the two umbrellas in Fig. 8.3—because of the difference in shape. ICP makes mistakes that EMKs don’t on examples that have different 3-D shape but similar 2-D shape. Differencing makes many of the same mistakes ICP scoring does, but fewer. This makes sense: differencing error is more or less ICP error with better handling of color, surface orientation and occlusion. An example of a false positive is shown in Fig. 8.5.

For the 64 segments in dataset B, it takes about 150 minutes to compute kernel descriptors and EMK features and 195 minutes to run matching (much of which is reading feature vectors from disk). ICP matching runs about 150 minutes (which depends linearly on the number of random ICP restarts), and scene differencing is another 75.

### 8.4.3 Object Discovery

Since all the similarities we calculate in sec. 8.3.3 are nonsymmetric, we have a choice of how to combine them when creating a similarity matrix. Given the scores for segment  $s$  against segment  $t$  and for  $t$  against  $s$ , we can set the similarity matrix entry to (the exponential of), for example, the mean, min or max of the two. These kernels all perform similarly when used with Normalized Cuts for our MRF inference, suggesting that while our similarities are not strictly symmetric, they are very close. In Fig. 8.4 we compare the performance of Normalized Cuts using the score from 2-D feature matching alone, the score from ICP alone and the combined score. Across the entire range of possible cluster counts for both datasets, combining the scores never decreases performance. Using the differencing-based score further improves clustering quality across all cluster counts. In Fig. 8.6 we show the clusters produced for dataset B with number of desired clusters set to 20 (results for other cluster numbers are similar). Even most of the textureless segments are correctly clustered, which could not have been accomplished using matching of 2-D features only. Furthermore, the approach is able to distinguish the different coffee mugs based on only slight color and shape differences.

In Table 8.3 we show the comparison of two leading spectral clustering algorithms, the Multiclass Normalized Cuts [196] and the Self-tuning Spectral Clustering [197]. We use the 48 segments in dataset A and manually label all pairs of segments as being the same object or not. Algorithms are evaluated using precision and recall of the binary labels on segment pairs. We vary the number of clusters from 2 to 48 and report the results with the best F-measures.

| algorithm                             | precision | recall | F-measure |
|---------------------------------------|-----------|--------|-----------|
| multiclass Ncuts [196]                | 0.939     | 0.986  | 0.962     |
| self-tuning spectral clustering [197] | 0.538     | 0.837  | 0.655     |

Table 8.3: comparison of spectral clustering algorithms. Precision-recall values and F-measures are computed using ground-truth same-object and different-object relations between segments. Normalized Cuts performs the best in all our experiments.



Figure 8.6: Object clusters produced for dataset B with number of desired clusters set to 20. Incorrect cluster members are outlined in red, and object IDs are given in small text. Clusters of size  $> 1$  are left of the line; singletons are to the right. (All three singletons are segments of the same object, a water bottle that the depth camera has trouble seeing due to translucency and refraction. The approach is able to distinguish between objects of very similar shape and appearance: coffee mugs 3, 8, and 10, and bowls 9, 12, and 14. While the EMK features are crucial to distinguish between individual mugs and bowls, 3D shape information via ICP is important to distinguish across these two classes. Some of the objects (2, 3) are oversegmented into multiple clusters.



### 8.5 *Summary and Future Work*

We have introduced an RGB-D object discovery system using multiple visits to a scene and shown that 3-D shape and motion information improve both segmentation and matching of 3-D segments. Our approach jointly analyzes multiple scenes to consistently segment objects that move between scenes. Segments are then combined into objects using graph-based clustering. Our experiments show that by combining shape and color descriptors, our approach can robustly detect objects and their motion between scenes, even when objects are textureless or have virtually the same shape as other objects, such as bowls or coffee mugs.

Our explorations into partial model matching have brought up interesting problems: the inside and outside of the same object, such as a mug, might have the same shape, but the surfaces have completely different orientations, making it hard to match or align them without knowledge of the fact that the object has a thin interior. Insides and outsides may also have different colors. It may also be possible to use information from some point clouds in a cluster to change others. For example, in cluster 11 of fig. 8.6, some clouds show the hole in the mug’s handle and some do not; we may be able to insert the hole into some clouds if it’s present in enough others.

RGB-D data have been available for only a short time, and the space of algorithms to fully use them has not been well explored. We expect to improve our system further in future by making more extensive use of the cues we have used here. In particular, while we currently use scene differencing for segmentation and alignment scoring, we would like to optimize this objective directly for alignment, rather than relying on ICP with its known sub-optimal objective. We now discuss preliminary investigations into how we might do this.

## Chapter 9

**SUMMARY AND FUTURE WORK****9.1 Motivation**

The tasks for which mobile robots promise to be useful range from relatively straightforward following, carrying and fetching tasks to entirely autonomous and very complicated operations such as natural resource extraction (on- or off-planet), maintenance of highly unpredictable systems like power plants, and the management of entire human lives (as in the case of sufferers from Alzheimer’s who are also physically disabled and so require a large amount of physical and mental assistance). At present, mobile robots don’t reliably have even the most basic capabilities for operating in unconstrained environments. Mapping, localization and navigation, prerequisites for any mobile work, are relatively well developed but are not yet robust enough for real use and are mostly designed for well-controlled environments. Our goal is to give mobile robots the minimal autonomy necessary to do any useful work in real-world environments populated by humans.

The problems we tackle are those of rigid object segmentation and modeling, necessities for working with movable objects. We make use of the connection between the informal notion of “objectness” and the formal notion of independence of movement, and use motion cues to identify objects. We focus on extending modeling and understanding of objects through time, as service robots in actual use will need to maintain environment models over long periods of time, ideally as long as decades. Real scenes robots encounter will contain many kinds of motion, and we emphasize using a variety of methods to deal with motion over short and long periods of time, while observed by the robot or not, whether caused by the robot, nearby humans, or other environmental processes.

**9.2 Contributions**

There is certainly previous work on the problem of segmenting objects using their motion. This thesis extends this field in two ways. Firstly, we compute motion cues in ways that allow for objects to be textureless, one major limitation of most previous work on object

segmentation. Secondly, we design and use a detailed and accurate probabilistic model of measurements from an RGB-D camera to make the best use of this new kind of sensor.

We present four algorithms for segmenting objects from motion cues. In sec. 4.3, we show how to use our probabilistic sensor measurement model to perform change detection in RGB-D scenes, and we spatially regularize change detection results to segment spatially connected clusters of surfaces that have moved over the same time intervals. This approach has limitations in the cases of very large and very small motions. In sec. 4.5 we use feature matching techniques similar to those currently popular in computer vision for all kinds of matching tasks, but extended to a dense representation rather than a sparse keypoint-based one, to estimate general motions of any objects over any time period. Due to the generality of this method, it is quite inefficient, so in sec. 5.3 we develop a segmentation method based on dense motion estimation over small periods of time. This method represents a step toward tracking and segmenting objects online during their movement. Finally, in sec. 5.4 we show how to segment an object from video of a simple human interaction with the object. This is a first step toward segmenting novel objects from *any* human interactions with the objects, complementing our other segmentation methods, which are not intended to process video of people.

In ch. 8 we show how to infer correspondences among segments obtained from any of these methods to estimate the movements of objects over more than two timesteps. We use a matching method based on 3-D point cloud alignment and our RGB-D change detection to produce a similarity metric for pairs of segments. In sec. 6.3 we discuss extending the use of change detection to not just matching of segments but also the registration process.

The methods of ch. 4 and ch. 8 mostly operate on pre-built 3-D maps. In ch. 6 we build on an *online* SLAM technique to segment rigid objects *during* rather than after scene reconstruction. We can then represent a scene with 3-D models of its movable and static parts *while mapping the scene for the first time*, and update these models online. In ch. 7 we add the ability to use the robot to improve online segmentation and modeling by choosing camera views to obtain, and show preliminary work on additionally using the robot’s manipulator to move objects while observing them. These capabilities—moving the camera and moving objects—are enough, when combined with the motion estimation and segmentation techniques we have introduced, to build a mobile robot that can enter a novel environment with no known objects, explore the scene, and segment all movable objects (that the sensor

can recognize—we still aren’t able to process transparent objects, for example) and build models of them. This represents some minimal level of competence necessary for a mobile robot working in human environments with human-designed objects. Further, we have made progress toward a system that can learn from whatever data are available, using video of human interactions with objects if any occur, video of its own interactions with objects if necessary, and otherwise, whatever video it happens to obtain, regardless of the robot’s level of control in moving the camera.

One interesting point is the degree to which we can build this complex system on the single operation of probabilistic change detection. We show in this work how to use change detection for segmentation of objects, for registration of 3-D segments (although it doesn’t yet work as well as the related but less sophisticated and GPU-based KinectFusion-style approach for this purpose), for registration of videos of different scenes to each other, and for choosing how to move the camera if segmentation is done actively and what manipulations to make if segmentation is done interactively. We consider the elegance of this approach to be a substantial point in its favor.

### **9.3 Future Directions**

This thesis discusses work toward the long-term goal of a basic level of robot autonomy in indoor environments designed for humans. There remain a large number of unsolved issues in building such a system, even abstracting away the low-level issue of intelligent motor control and the very high-level issue of avoiding antisocial behavior patterns. We now discuss some thoughts on missing pieces in this system and on integration of our existing work into such a system.

#### *9.3.1 Making Use of Visual Point Feature Matching*

Visual point feature matching (as exemplified by the work of Gordon and Lowe [65]) is currently the most popular paradigm for matching in computer vision. For purposes of original research, we have avoided using visual point feature matching as much as possible in this thesis, preferring dense pixel-based approaches. However, practical systems can make good use of point feature matching to improve both efficiency and accuracy. For example, our group’s RGB-D Mapping SLAM system [75], which we discuss in sec. 2.2, uses visual point

feature matching to initialize frame-to-frame alignment in the visual odometry step because it is both more efficient than ICP and useful in preventing ICP from diverging on some frames. The Large-Displacement Optical Flow of Brox and Malik [25] incorporates a sparse descriptor-matching term into variational optical flow (with a formulation similar to ours at a high level) in order to both speed up convergence when motion is large and at times change the basin of convergence in which the solution lies. Our own experience combining sparse descriptor matching with a dense iterative optimization for alignment between different scenes, discussed in sec. 6.3, is also that a combination of the two approaches performs better than either individually. One avenue of improvement to the performance of our motion estimation algorithms, therefore, is to incorporate information from sparse descriptor matching, preferably in the formulation of each algorithm rather than as a separate initialization. Additionally, we could modify the seed selection step of our dense feature matching approach to make highly distinctive keypoints more likely than others to be selected as seeds, with the idea that our motion estimation at those points would be at least slightly more accurate than in less textured regions.

### 9.3.2 *Making Use of Video Containing People*

We have demonstrated object segmentation from video of a restricted human interaction with an object, but this requires that the person focus entirely on helping the robot for at least a short period of time. This is a useful tool as it results in near-perfect segmentations, but a real-world system will need to be able to process more general video of people interacting with objects for the people’s purposes rather than for the robot’s. Segmenting people in natural scenes is an extremely hard problem, has been a research area for decades, and is commonly avoided in research papers by using special colored clothing (e.g. [36]) or depth discontinuity information (e.g. [162]). Recent work on tracking robot manipulators using articulation-aware algorithms [109] and on tracking people with a focus on outlines rather than accuracy of motion estimation [155] makes us optimistic that people can be tracked to some degree, allowing for segmentation of people from the objects they are manipulating. Nevertheless, this direction is less well-defined than the others we discuss.

### 9.3.3 *Intelligent Manipulation*

As is evident from sec. 7.3, intelligently choosing where and how to touch objects to cause them to move is still an open problem. Hager and Wegbreit [72] propose a promising occlusion-ordering-based approach to object modeling in novel scenes, in which clusters representing objects are first identified and then processed in front-to-back order from the camera’s point of view. One advantage to this approach that is not discussed in that paper is that if segmentation is uncertain, processing the easiest-to-see objects first is likely to be the best way to improve initial estimates of the segmentation of future objects to be processed. The major disadvantage to this approach, however, is that an approximate segmentation must be produced before object selection can begin. We have seen how imperfect static bottom-up segmentation is, although it could be improved by the use of object detection techniques.

### 9.3.4 *Integrating Information from Many Segmentation Methods*

We have discussed a variety of object segmentation algorithms: change detection on static scenes, feature matching, dense motion estimation, and processing of intentional demonstrations by a human. Each gives us slightly different information. Segmentation with change detection accumulates evidence for two labels, changed and not-changed, at each pixel in each video frame or 3-D map we use for change detection. The iterative motion fitting steps of our feature matching algorithm and our algorithm based on dense motion estimation both create a set of labels each scene element can take, corresponding to different motions discovered. One result of processing a segmentation demonstration is a map of pixels in the demonstration start and end frames to a Boolean value giving whether the pixel is part of the object being demonstrated. Each of these algorithms provides us with a set of labels that help distinguish objects from one another and the background, as well as a measure of each scene element’s agreement with each label. Most of these algorithms already spatially regularize label assignments by running inference in a Markov random field with nodes at scenels (scene elements: 3-D points, surfels, mesh vertices, or similar) and edges between spatially proximate scenels. If we could define a similar Markov random field whose structure changes over time to accommodate the changing locations of surfaces as scene reconstruction progresses, we could aggregate at each scenel the labels produced by different

runs of various two-time-point segmentation algorithms over different time intervals. There are many algorithmic and efficiency issues to be worked out for such a method to be useful, especially in how to reduce the label set size periodically since it grows over time, but this is a promising approach to combining information from different segmentation methods over a long time duration.

### *9.3.5 Life-Long Object Discovery*

Our object discovery technique, along with most others, is an off-line algorithm, requiring to be rerun in full each time new data need to be added to the system. Work on online object discovery does exist (e.g., [130] uses shape matching, albeit less sophisticated than ours, over both short and long time intervals), but no such work has demonstrated use over long periods of time and on large numbers of objects with similar appearance. Finman et al. [55] present first steps toward online probabilistic object modeling, but they use very coarse features that will not work on large sets of objects, and some of their clustering is done manually. We would use an approach based on a pairwise similarity function defined by using our dense change-detection-based 3-D segment alignment and scoring methods, with similarity thresholds for creating clusters of segments. This would allow each object model to be both completed and improved in accuracy slowly over time, eventually becoming complete and not needing to be updated. Zillich et al.’s criterion for stopping improvement of an object model [198] is a good start; we would adapt it to the non-sparse-feature-based case by replacing the use of feature points with something like the set of vertices on a mesh made from a volumetric representation of a model.

## BIBLIOGRAPHY

- [1] Second IEEE workshop on egocentric vision at CVPR 2012. <http://egovision12.cc.gatech.edu/>.
- [2] SICK AG. Sick outdoor laser measurement technology. [http://www.sick.com/group/EN/home/products/product\\_portfolio/laser\\_measurement\\_systems/Pages/outdoor\\_laser\\_measurement\\_technology.aspx](http://www.sick.com/group/EN/home/products/product_portfolio/laser_measurement_systems/Pages/outdoor_laser_measurement_technology.aspx).
- [3] D. Aiger, N. Mitra, and D. Cohen-Or. 4-points congruent sets for robust pairwise surface registration. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 2008.
- [4] P. Alimi, D. Meger, and J. Little. Object persistence in 3-d for home robots.
- [5] H. Andreasson and A. Lilienthal. 6d scan registration using depth-interpolated local image features. *Robotics and Autonomous Systems*, 2010.
- [6] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative learning of markov random fields for segmentation of 3-d scan data. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [7] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. From contours to regions: An empirical evaluation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [8] N. Atanasov, B. Sankaran, J. Ny, T. Koletschka, G. Pappas, and K. Daniilidis. A hypothesis testing framework for active object detection. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
- [9] A. Aydemir and P. Jensfelt. Exploiting and modeling local 3-d structure for predicting object locations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.
- [10] R. Bajcsy. Active perception. Technical report, Penn, 1988.
- [11] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
- [12] Barrett, Inc. Wam<sup>TM</sup> arm. <http://www.barrett.com/robot/products-arm.htm>.



- [13] N. Bergstrom, M. Bjorkman, and D. Kragic. Generating object hypotheses in natural scenes through human-robot interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.
- [14] C. Bersch, D. Pangercic, S. Osentoski, K. Maroney, and M. Beetz. Segmentation of cluttered scenes through interactive perception. In *ICRA Workshop on Semantic Perception and Mapping for Knowledge-enabled Service Robotics*, 2012.
- [15] P. Besl and N. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 1992.
- [16] P. Bhat, K. Zheng, N. Snavely, A. Agarwala, M. Agrawala, M. Cohen, and B. Curless. Piecewise image registration in the presence of large motions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.
- [17] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.
- [18] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun. Towards object mapping in dynamic environments with mobile robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.
- [19] N. Blodow, L. Goron, Z. Marton, D. Pangercic, T. Ruehr, M. Tenorth, and M. Beetz. Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.
- [20] L. Bo, X. Ren, and D. Fox. Kernel descriptors for visual recognition. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.
- [21] L. Bo and C. Sminchisescu. Efficient match kernels between sets of features for visual recognition. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.
- [22] J. Boyd. Robotic construction machine causes explosion at fukushima. <http://spectrum.ieee.org/automaton/robotics/industrial-robots/robotic-construction-machine-causes-explosion-at-fukushima>.
- [23] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2001.
- [24] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High-accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision (ECCV)*, 2004.

- [25] T. Brox and J. Malik. Large-displacement optical flow: Descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2010.
- [26] T. Brox and J. Malik. Object segmentation by long-term analysis of point trajectories. In *European Conference on Computer Vision (ECCV)*, 2010.
- [27] A. Bruhn and J. Weickert. A confidence measure for variational optical flow methods. Technical Report 106, Saarland University Mathematics Department, 2004.
- [28] A. Bruhn, J. Weickert, and C. Schnoerr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International Journal of Computer Vision*, 2005.
- [29] M. Calonder, V. Lepetit, and P. Fua. Keypoint signatures for fast learning and recognition. In *European Conference on Computer Vision (ECCV)*, 2008.
- [30] J. Carreira and C. Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2012.
- [31] K. Chang, T. Liu, and S. Lai. From co-saliency to co-segmentation: An efficient and fully unsupervised energy minimization model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.
- [32] L. Chang, J. Smith, and D. Fox. Interactive singulation of objects from a pile. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [33] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. *Image and Vision Computing*, 1992.
- [34] P. Cignoni. Meshlab. <http://meshlab.sourceforge.net/>.
- [35] A. Colburn, A. Agarwala, A. Hertzmann, B. Curless, and M. Cohen. Image-based remodeling. *IEEE Transactions on Visualization and Computer Graphics*, 2014.
- [36] J. Cole, D. Grimes, and R. Rao. Learning full-body motions from monocular vision: Dynamic imitation in a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.
- [37] A. Collet, S. Srinivasa, and M. Hebert. Structure discovery in multi-modal data : a region-based approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

- [38] A. Collet, M. Torres, and S. Srinivasa. The moped framework: Object recognition and pose estimation for manipulation. *International Journal of Robotics Research (IJRR)*, 2011.
- [39] A. Collet Romea, D. Berenson, S. Srinivasa, and D. Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- [40] C. Connolly. The determination of next best views. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1985.
- [41] Intel Corporation. Creative(tm) interactive gesture camera developer kit. [http://click.intel.com/intelsdk/Creative\\_Interactive\\_Gesture\\_Camera\\_Developer\\_Kit-P2061.aspx](http://click.intel.com/intelsdk/Creative_Interactive_Gesture_Camera_Developer_Kit-P2061.aspx).
- [42] LeapMotion corporation. Leap. <https://leapmotion.com/>.
- [43] Mesa Imaging corporation. Swissranger sr4000. <http://www.mesa-imaging.ch/prodview4k.php>.
- [44] Point Grey Research corporation. Bumblebee 2 ccd camera. [http://www.ptgrey.com/products/bumblebee2/bumblebee2\\_stereo\\_camera.asp](http://www.ptgrey.com/products/bumblebee2/bumblebee2_stereo_camera.asp).
- [45] PrimeSense corporation. Primesense. <http://www.primesense.com/>.
- [46] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 1996.
- [47] S. Dasgupta. Online and streaming algorithms for clustering. <http://cseweb.ucsd.edu/~dasgupta/291/lec6.pdf>.
- [48] I. Dhillon, Y. Guan, and B. Kulis. Kernel k-means, spectral clustering and normalized cuts. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.
- [49] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon Robotics Institute, 2010.
- [50] M. Dogar and S. Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, 2013.
- [51] H. Du, P. Henry, X. Ren, M. Cheng, D. Goldman, S. Seitz, and D. Fox. Interactive 3d modeling of indoor environments with a consumer depth camera. In *International Conference on Ubiquitous Computing (UbiComp)*, 2011.

- [52] O. Enqvist, K. Josephson, and F. Kahl. Optimal correspondences from pairwise constraints. In *International Conference on Computer Vision (ICCV)*, 2009.
- [53] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge 2011 detection task. [http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/workshop/voc\\_det.pdf](http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/workshop/voc_det.pdf).
- [54] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge 2011 segmentation task. [http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/workshop/voc\\_seg.pdf](http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/workshop/voc_seg.pdf).
- [55] R. Finman, T. Whelan, M. Kaess, and J. Leonard. Toward life-long object segmentation from change detection in dense rgb-d maps. In *European Conference on Mobile Robots*, 2013.
- [56] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981.
- [57] P. Fitzpatrick. First contact: an active vision approach to segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.
- [58] D. Forsyth, J. Malik, M. Fleck, H. Greenspan, T. Leung, S. Belongie, C. Carson, and C. Bregler. *Finding pictures of objects in large collections of images*. 1996.
- [59] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research (JAIR)*, 1999.
- [60] D. Fox, W. Burgard, and S. Thrun. Markov localization for reliable robot navigation and people detection. In *Modelling and Planning for Sensor-Based Intelligent Robot Systems*, Lecture Notes in Computer Science. Springer Verlag, 1999.
- [61] B. Glocker, N. Paragios, N. Komodakis, G. Tziritas, and N. Navab. Optical flow estimation with uncertainties through dynamic mrfs. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.
- [62] C. Goldfeder, M. Ciocarlie, H. Dang, and P. Allen. The columbia grasp database. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- [63] D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research (JAIR)*, 2011.

- [64] H. Gonzalez-Baños and J. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research (IJRR)*, 2002.
- [65] I. Gordon and D. Lowe. What and where: 3-d object recognition with accurate pose. In *Toward Category-Level Object Recognition*. 2006.
- [66] J. Gottfried, J. Fehr, and C. Garbe. Computing range flow from multimodal kinect data. In *International Symposium on Visual Computing (ISVC)*, 2011.
- [67] S. Goza, R. Ambrose, M. Diftler, and I. Spain. Telepresence control of the nasa/darpa robonaut on a mobility platform. In *Conference on Human Factors in Computing Systems (CHI)*, 2004.
- [68] K. Grauman and T. Darrell. Unsupervised learning of categories from sets of partially matching image features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.
- [69] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems (RSS)*, 2007.
- [70] S. Hadfield and R. Bowden. Kinecting the dots: Particle-based scene flow from depth sensors. In *International Conference on Computer Vision (ICCV)*, 2011.
- [71] D. Haehnel, R. Triebel, W. Burgard, and S. Thrun. Map building with mobile robots in dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.
- [72] G. Hager and B. Wegbreit. Scene parsing using a prior world model. *International Journal of Robotics Research (IJRR)*, 2011.
- [73] P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch volumes: Segmentation-based consistent mapping with rgb-d cameras. In *International Conference on 3-D Vision (3DV)*, 2013.
- [74] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *International Symposium on Experimental Robotics (ISER)*, 2010.
- [75] 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. *International Journal of Robotics Research (IJRR)*, 2012.

- [76] E. Herbst, P. Henry, X. Ren, and D. Fox. Toward object discovery and modeling via 3-d scene comparison. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
- [77] E. Herbst, X. Ren, and D. Fox. Rgb-d object discovery via multi-scene analysis. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.
- [78] E. Herbst, X. Ren, and D. Fox. Object segmentation from motion with dense feature matching. In *ICRA Workshop on Semantic Perception, Mapping and Exploration*, 2012.
- [79] E. Herbst, X. Ren, and D. Fox. Rgb-d flow: Dense 3-d motion estimation using color and depth. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
- [80] T. Hermans, J. Rehg, and A. Bobick. Guided pushing for object singulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.
- [81] G. Hollinger, B. Englot, F. Hover, U. Mitra, and G. Sukhatme. Uncertainty-driven view planning for underwater inspection. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [82] D. Holz, N. Basilico, F. Amigoni, and S. Behnke. Evaluating the efficiency of frontier-based exploration strategies. In *41st International Symposium on Robotics and 6th German Conference on Robotics (ISR/ROBOTIK)*, 2010.
- [83] D. Holz, M. Nieuwenhuisen, D. Droeschel, J. Steckler, A. Berner, J. Li, R. Klein, and S. Behnke. Active recognition and manipulation for mobile robot bin picking. In *Gearing up and accelerating cross-fertilization between academic and industrial robotics research in Europe - Technology transfer experiments from the ECHORD project (v94 of Springer Tracts in Advanced Robotics)*. 2014.
- [84] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society A*, 1987.
- [85] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 1981.
- [86] A. Hornung, M. Phillips, E. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [87] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: an efficient probabilistic 3-d mapping framework based on octrees. *Autonomous Robots*, 2013.

- [88] E. Hsiao, A. Collet, and M. Hebert. Making specific features less discriminative to improve point-based 3-d object recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.
- [89] E. Hsiao and M. Hebert. Occlusion reasoning for object detection under arbitrary viewpoint. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [90] F. Huguet and F. Devernay. A variational method for scene flow estimation from stereo sequences. In *International Conference on Computer Vision (ICCV)*, 2007.
- [91] S. Javdani, M. Klingensmith, J. Bagnell, N. Pollard, and S. Srinivasa. Efficient touch-based localization through submodularity. Technical Report CMU-RI-TR-12-25, CMU Robotics Institute, 2012.
- [92] A. Johnson. *Spin Images: a Representation for 3-D Surface Matching*. PhD thesis, Carnegie Mellon University, 1997.
- [93] A. Joulin, F. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.
- [94] R. Kaestner, S. Thrun, M. Montemerlo, and M. Whalley. A non-rigid approach to scan alignment and change detection using range sensor data. In *Symposium on Field and Service Robotics*, 2005.
- [95] H. Kang, M. Hebert, and T. Kanade. Discovering object instances from scenes of daily living. In *International Conference on Computer Vision (ICCV)*, 2011.
- [96] D. Katz and O. Brock. Interactive segmentation of articulated objects in 3-d. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
- [97] D. Katz, M. Kazemi, J. Bagnell, and A. Stentz. Semi-autonomous manipulation of natural objects. Technical report, Carnegie Mellon, 2012.
- [98] D. Katz, M. Kazemi, J. Bagnell, and A. Stentz. Interactive segmentation, tracking, and kinematic modeling of unknown 3-d articulated objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
- [99] J. Kenney, T. Buckley, and O. Brock. Interactive segmentation for manipulation in unstructured environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

- [100] G. Kim, C. Faloutsos, and M. Hebert. Unsupervised modeling of object categories using link analysis techniques. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.
- [101] Y. Kim, C. Theobalt, J. Diebel, J. Kosecka, B. Matusik, and S. Thrun. Multi-view image and tof sensor fusion for dense 3-d reconstruction. In *3dim*, 2009.
- [102] T. Kollar and N. Roy. Utilizing object-object and object-scene context when planning to find things. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- [103] C. Kondermann, R. Mester, and C. Garbe. A statistical confidence measure for optical flows. In *European Conference on Computer Vision (ECCV)*, 2008.
- [104] K. Konolige. Sparse sparse bundle adjustment. In *British Machine Vision Conference (BMVC)*, 2010.
- [105] K. Konolige and J. Bowman. Towards life-long visual maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.
- [106] K. Konolige and P. Mihelich. Technical description of kinect calibration. [http://wiki.ros.org/kinect\\_calibration/technical/](http://wiki.ros.org/kinect_calibration/technical/).
- [107] H. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.
- [108] H. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *Robotics: Science and Systems (RSS)*, 2013.
- [109] M. Krainin, B. Curless, and D. Fox. Autonomous generation of complete 3-d object models using next-best-view manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
- [110] M. Krainin, P. Henry, X. Ren, and D. Fox. Manipulator and object tracking for in hand 3d object modeling. *International Journal of Robotics Research (IJRR)*, 30(9), 2011.
- [111] M. Krainin, K. Konolige, and D. Fox. Exploiting segmentation for robust 3-d object matching. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [112] H. Kress-Gazit. Robot challenges: Toward development of verification and synthesis techniques. *Robotics and Automation Magazine*, 2011.



- [113] K. Lai, L. Bo, X. Ren, and D. Fox. A scalable tree-based approach for joint object and pose recognition. In *National Conference on Artificial Intelligence (AAAI)*, 2011.
- [114] K. Lai and D. Fox. Object recognition in 3D point clouds using web data and domain adaptation. *International Journal of Robotics Research (IJRR)*, 29(8), 2010.
- [115] S. LaValle. Rapidly exploring random trees: a new tool for path planning. Technical Report TR 98-11, Iowa State, 1998.
- [116] A. Letouzey, B. Petit, and E. Boyer. Scene flow from depth and color images. In *British Machine Vision Conference (BMVC)*, 2011.
- [117] Velodyne LIDAR. Velodyne lidar. <http://velodynelidar.com/lidar/lidar.aspx>.
- [118] C. Liu, W. Freeman, E. Adelson, and Y. Weiss. Human-assisted motion annotation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.
- [119] D. Liu and T. Chen. A topic-motion model for unsupervised video object discovery. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
- [120] W. Lorensen and H. Cline. Marching cubes: A high-resolution 3-d surface construction algorithm. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 1987.
- [121] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.
- [122] Hokuyo Automatic Co. Ltd. Hokuyo photo sensor. <http://www.hokuyo-aut.jp/02sensor/index.html#scanner>.
- [123] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Image Understanding Workshop*, 1981.
- [124] T. Lukins and R. Fisher. Colour-constrained 4-d flow. In *British Machine Vision Conference (BMVC)*, 2005.
- [125] J. MacCormick. How does the kinect work? <http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf>.
- [126] T. Malisiewicz and A. Efros. Improving spatial support for objects via multiple segmentations. In *Proc. of the British Machine Vision Conference*, 2007.

- [127] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. 2004.
- [128] J. Mason, B. Marthi, and R. Parr. Object disappearance for object discovery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.
- [129] A. Mishra, A. Srivastava, and Y. Aloimonos. Segmenting "simple" objects using rgb-d. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [130] J. Modayil and B. Kuipers. Bootstrap learning for object discovery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.
- [131] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 1957.
- [132] H. Nagel and W. Enkelmann. An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 1986.
- [133] O. Nestares, D. Fleet, and D. Heeger. Likelihood functions and confidence bounds for total-least-squares problems. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2000.
- [134] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. 2011.
- [135] C. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3-d reconstruction and tracking. In *International Conference on 3-D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, 2012.
- [136] NVIDIA Corporation. Cusp. <https://developer.nvidia.com/cusp/>.
- [137] E. Olson, M. Walter, S. Teller, and J. Leonard. Single-cluster spectral graph partitioning for robotics applications. In *Robotics: Science and Systems (RSS)*, 2005.
- [138] L. Ott and F. Ramos. Unsupervised incremental learning for long-term autonomy. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [139] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 2000.
- [140] C. Potthast and G. Sukhatme. A probabilistic framework for next best view estimation in a cluttered environment. *Journal of Visual Communication and Image Representation (JVCI)*, 2013.

- [141] A. Pronobis. *Semantic Mapping with Mobile Robots*. PhD thesis, KTH, 2011.
- [142] S. Pundlik and S. Birchfield. Motion segmentation at any speed. In *British Machine Vision Conference (BMVC)*, 2006.
- [143] J. Quiroga, F. Devernay, and J. Crowley. Scene flow by tracking in intensity and depth data. In *CVPR Workshops*, 2012.
- [144] F. Ramos, D. Fox, and H. Durrant-Whyte. Crf-matching: Conditional random fields for feature-based scan matching. In *Robotics: Science and Systems (RSS)*, 2007.
- [145] X. Ren and L. Bo. Discriminatively trained sparse code gradients for contour detection. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.
- [146] X. Ren and M. Philipose. Egocentric recognition of handled objects: Benchmark and analysis. In *Proc. EgoVision Workshop*, 2009.
- [147] M. Ross and L. Kaelbling. Learning static object segmentation from motion segmentation. In *National Conference on Artificial Intelligence (AAAI)*, 2005.
- [148] E. Rosten and T. Drummond. Fusing points and lines for high-performance tracking. In *International Conference on Computer Vision (ICCV)*, 2005.
- [149] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV)*, 2006.
- [150] A. Roussos, C. Russell, R. Garg, and L. Agapito. Dense multibody motion estimation and reconstruction from a handheld camera. 2012.
- [151] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of 3d object models from partial views. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- [152] B. Russell, A. Efros, J. Sivic, W. Freeman, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.
- [153] R. Rusu, N. Blodow, Z. Marton, and M. Beetz. Close-range scene segmentation and reconstruction of 3-d point cloud maps for mobile manipulation in domestic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.
- [154] R. Salas-Moreno, R. Newcombe, H. Strasdat, P. Kelly, and A. Davison. Slam++: Simultaneous localization and mapping at the level of objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

- [155] P. Sand and S. Teller. Particle video. *International Journal of Computer Vision*, 2008.
- [156] B. Sapp, A. Saxena, and A. Ng. A fast data collection and augmentation procedure for object recognition. In *National Conference on Artificial Intelligence (AAAI)*, 2008.
- [157] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 2007.
- [158] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems (RSS)*, 2013.
- [159] J. Schulman, A. Lee, J. Ho, and P. Abbeel. Tracking deformable objects with point clouds. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
- [160] S. Shen, N. Michael, and V. Kumar. Autonomous indoor 3-d exploration with a micro-aerial vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [161] J. Shin, R. Triebel, and R. Siegwart. Unsupervised discovery of repetitive objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.
- [162] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, page 3, 2011.
- [163] N. Silberman, P. Kohli, D. Hoiem, and R. Fergus. Indoor segmentation and support inference from rgb-d images. In *European Conference on Computer Vision (ECCV)*, 2012.
- [164] T. Southey and J. Little. Object discovery through motion, appearance and shape. In *AAAI Workshop on Cognitive Robotics*, 2006.
- [165] H. Spies, B. J. ahne, and J. Barron. Range flow estimation. *Computer Vision and Image Understanding*, 2002.
- [166] A. Stein, D. Hoiem, and M. Hebert. Learning to find object boundaries using motion cues. In *International Conference on Computer Vision (ICCV)*, 2007.
- [167] H. Strasdat, J. Montiel, and A. Davison. Real-time monocular slam: Why filter? In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

- [168] J. Sturm, V. Pradeep, C. Stachniss, C. Plagemann, K. Konolige, and W. Burgard. Learning kinematic models for articulated objects. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.
- [169] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.
- [170] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 1998.
- [171] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [172] S. Thrun and M. Montemerlo. The graphslam algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research (IJRR)*, 2005.
- [173] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical report, Carnegie Mellon, #CMU-CS-91-132, 1991.
- [174] R. Triebel, B. Frank, J. Meyer, and W. Burgard. First steps towards a robotic system for flexible volumetric mapping of indoor environments. In *IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.
- [175] R. Triebel, J. Shin, and R. Siegwart. Segmentation and unsupervised part-based discovery of repetitive objects. In *Robotics: Science and Systems (RSS)*, 2010.
- [176] A. Ude, D. Schiebener, N. Sugimoto, and J. Morimoto. Integrating surface-based hypotheses and manipulation for autonomous segmentation and learning of object representations. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.
- [177] J. van de Ven, F. Ramos, and G. Tipaldi. An integrated probabilistic model for scan matching, moving object detection and motion estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.
- [178] H. van Hoof, O. Kroemer, H. Ben Amor, and J. Peters. Maximally informative interaction learning for scene exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.
- [179] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. In *International Conference on Computer Vision (ICCV)*, 1999.
- [180] S. Vicente, C. Rother, and V. Kolmogorov. Object cosegmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

- [181] C. Vogel, K. Schindler, and S. Roth. 3-d scene flow estimation with a rigid motion prior. In *International Conference on Computer Vision (ICCV)*, 2011.
- [182] C. Wang and C. Thorpe. Simultaneous localization and mapping with detection and tracking of moving objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.
- [183] A. Wedel, T. Brox, T. Vaudrey, C. Rabe, U. Franke, and D. Cremers. Stereoscopic scene flow computation for 3d motion understanding. *International Journal of Computer Vision*, 2011.
- [184] A. Wedel, D. Cremers, T. Pock, and H. Bischof. Structure- and motion-adaptive regularization for high-accuracy optic flow. In *International Conference on Computer Vision (ICCV)*, 2009.
- [185] A. Wedel, A. Meissner, C. Rabe, U. Franke, and D. Cremers. Detection and segmentation of independently moving objects from dense scene flow. In *International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, 2009.
- [186] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic huber- $l_1$  optical flow. In *British Machine Vision Conference (BMVC)*, 2009.
- [187] J. Wettach and K. Berns. Dynamic frontier-based exploration with a mobile indoor robot. In *41st International Symposium on Robotics and 6th German Conference on Robotics (ISR/ROBOTIK)*, 2010.
- [188] T. Whelan, M. Kaess, J. Leonard, and J. McDonald. Deformation-based loop closure for large scale dense rgb-d slam. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.
- [189] Wikipedia. Canesta. <http://en.wikipedia.org/wiki/Canesta>.
- [190] Wikipedia. Time-of-flight camera. [http://en.wikipedia.org/wiki/Time-of-flight\\_camera](http://en.wikipedia.org/wiki/Time-of-flight_camera).
- [191] O. Williams and A. Fitzgibbon. Gaussian process implicit surfaces. In *Gaussian Processes in Practice*. 2007.
- [192] A. Witkin. Scale-space filtering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1983.

- [193] D. Wolf and G. Sukhatme. Online simultaneous localization and mapping in dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.
- [194] K. Wurm, D. Hennes, D. Holz, R. Rusu, C. Stachniss, K. Konolige, and W. Burgard. Hierarchies of octrees for efficient 3-d mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.
- [195] B. Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence, Robotics and Automation*, 1997.
- [196] Stella X. Yu and Jianbo Shi. Multiclass spectral clustering. In *International Conference on Computer Vision (ICCV)*, 2003.
- [197] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.
- [198] M. Zillich, J. Prankl, T. Moerwald, and M. Vincze. Knowing your limits: Self-evaluation and prediction in object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.
- [199] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *International Journal of Robotics Research (IJRR)*, 2013.

## Appendix A

### PSEUDOCODE FOR RGB-D SENSOR MEASUREMENT MODELS

#### A.1 Accurate Model

Our accurate pixelwise RGB-D noise model based on that of Fox et al. [59].

$\triangleright c_d$ , from [59], is the probability of the beam hitting a surface at depth  $l$  given there is a surface there and the beam hasn't hit before  $l$   
 $\triangleright c_r$ , from [59], is the probability of there being a surface at depth  $l$  along a given beam (different  $l$  values are treated as independent)  
 $\triangleright p_{bias}$ , from [59], is the fraction of total probability to be allocated to the random-noise part of the model  
 $\triangleright$  return: tables `probDXGivenMoved`, `probDXGivenNotMoved`, `probNotHitGivenDXAndNotMoved`

```

1: function PRECOMPUTE
2:    $expZBinSize \leftarrow .005$   $\triangleright$  size of quantization bins for expected depth (affects table size in memory)
3:    $psdepths \leftarrow$  a list of the unique depths the depth sensor can return, in mm
4:    $numExpZBins \leftarrow maxZ/expZBinSize$ 
5:    $numObsZBins \leftarrow psdepths.size$ 

 $\triangleright$  we'll produce a table for each of  $numSigmaBins$  values of depth uncertainty sampled logarithmically between these limits, and later quantize the actual expected-depth uncertainty values into these bins
6:    $(minSigma, maxSigma) \leftarrow 1e-3, 3$ 
7:   for  $k \leftarrow 0..numSigmaBins - 1$  do
8:      $sigmaBinBounds[k] \leftarrow e^{\log(minSigma) + (\log(maxSigma) - \log(minSigma)) \times \frac{k}{numSigmaBins}}$ 

9:   for  $i \leftarrow 0..numSigmaBins - 1$  do
 $\triangleright$  get range of each depth bin, in mm
10:    for  $n \leftarrow 0..numObsZBins - 1$  do
11:       $minDepthForObsZBin[n] \leftarrow$  the minimum depth that the sensor quantizes to this bin (will be  $> minDepthForObsZBin[n - 1]$ )
12:       $maxDepthForObsZBin[n] \leftarrow$  the maximum depth that the sensor quantizes to this bin

13:    $probHitObjAtLGivenObjAtLAndNoHitBeforeL \leftarrow c_d$ 
14:   for  $k \leftarrow firstSigmaBin..lastSigmaBin$  do
15:      $probHitBeforeLGivenMoved \leftarrow 0$ 
16:     for  $l \leftarrow 0..numExpZBins - 1$  do  $\triangleright$  depth of surface we might hit
17:        $z \leftarrow (l + .5) \times expZBinSize$   $\triangleright$  middle of expected-depth bin
 $\triangleright$  hitDist:  $p(\text{reading is } z_{obs} \text{ — the object hit is at depth } z)$ 
18:        $\sigma_d \leftarrow \frac{sigmaBinBounds[k] + sigmaBinBounds[k+1]}{2}$   $\triangleright$  halfwidth of expected-depth bin

```



```

19:      sum ← 0
20:      for n ← 0..numObsZBins - 1 do
21:          firstSameIndexDepth ← .001 × minDepthForObsZBin[n]
22:          lastSameIndexDepth ← .001 × maxDepthForObsZBin[n]
23:          probDXGivenHitL[n] =  $\int_{y=firstSameIndexDepth-.0005}^{lastSameIndexDepth+.0005} \mathcal{N}(y; z, \sigma_d) \leftarrow$ 
24:          sum ← sum + probDXGivenHitL[n]

    ▷ calculate tables for the case when the expected surface has moved
25:      probObjAtLGivenMoved ← cr
26:      probHitLGivenMoved ← (1 - probHitBeforeLGivenMoved) * probObjAtLGivenMoved *
    probHitObjAtLGivenObjAtLAndNoHitBeforeL
27:      for j ← 0..numObsZBins - 1 do
28:          probDXGivenMoved[k][j] ← probDXGivenMoved[k][j] + probHitLGivenMoved *
    probDXGivenHitL[j]
29:      probHitBeforeLGivenMoved ← probHitBeforeLGivenMoved + probHitLGivenMoved

    ▷ calculate tables for the case when the expected surface hasn't moved
    ▷ we'll here set probDXGivenNotMoved to  $p(z, h = 1 | m = 0)$ , and fix it later to match its name
30:      for i ← 0..numExpZBins do
31:          probObjAtLGivenNotMoved ← if (l == i) then 1 else cr
32:          probHitLGivenNotMoved ← (1 - probHitBeforeLGivenNotMoved[i]) ×
    probObjAtLGivenNotMoved × probHitObjAtLGivenObjAtLAndNoHitBeforeL
33:      for j ← 0..numObsZBins - 1 do
34:          deltaProb ← probHitLGivenNotMoved × probDXGivenHitL[j]
35:          if i ≠ l then
36:              probDXAndNotHitGivenNotMoved[k][i][j] ← probDXAndNotHitGivenNotMoved[k][i][j] +
    deltaProb    ▷ hit is true if the z bins of the measurement-causing and expected surfaces are equal
37:          else
38:              probDXGivenNotMoved[k][i][j] ← probDXGivenNotMoved[k][i][j] + deltaProb
39:              probHitBeforeLGivenNotMoved[i] ← probHitBeforeLGivenNotMoved[i] +
    probHitLGivenNotMoved

    ▷ add random noise (a component that's uniform over z)
40:      for j ← 0..numObsZBins - 1 do
41:          binSize ← .001 × (maxDepthForObsZBin[j] - minDepthForObsZBin[j] + 1)
42:          biasAtDepth ← pbias ×  $\frac{binSize}{maxZ}$ 
43:          probDXGivenMoved[k][j] ← biasAtDepth + (1 - pbias_weight) × probDXGivenMoved[k][j]

    ▷ finalize tables for the case when the expected surface hasn't moved
44:      for i ← 0..numExpZBins do
45:          for j ← 0..numObsZBins - 1 do
46:              binSize = .001 × (maxDepthForObsZBin[j] - minDepthForObsZBin[j] + 1) ←
47:              biasAtDepth ← pbias ×  $\frac{binSize}{maxZ}$ 
48:              probDXAndNotHitGivenNotMoved[k][i][j] ← biasAtDepth + (1 - pbias) ×

```

```

    probDXAndNotHitGivenNotMoved[k][i][j]
49:      probDXGivenNotMoved[k][i][j] ← (1 - pbias) × probDXGivenNotMoved[k][i][j] +
    probDXAndNotHitGivenNotMoved[k][i][j]
50:      probNotHitGivenDXAndNotMoved[k][i][j] ←  $\frac{probDXAndNotHitGivenNotMoved[k][i][j]}{probDXGivenNotMoved[k][i][j]}$ 

```

▷ some constants for the surface normal component

▷ these kappas are the von Mises-Fisher concentration parameters, set by the user

```

51:    pnormal_moved_pdfval ← pnormal_moved_kappa / (4π × sinh(pnormal_moved_kappa))    ▷ μ = 0, so von
    Mises-Fisher normalizes reduces to constant
52:    logPhit_pdfval_precalc ← log(pnormal_hit_kappa) - log(4π × sinh(pnormal_hit_kappa))

```

▷ compute  $p(z \mid m = 1)$  and  $p(z \mid m = 0)$

▷ args expZ, expZSigma, expRGB, expNormal: expected surface attributes, including depth standard deviation

▷ args obsZ, obsRGB, obsNormal: observed surface attributes

▷ return: prob{D,C,N}{M,NM}: probabilities of observations in {depth, color, normal} components given that the surface has {moved, not moved}

```

53: function COMPUTE(expZ, expZSigma, obsZ, expRGB, obsRGB, expNormal, obsNormal)
54:   zExpBin ← getExpZBin(expZ)    ▷ quantize depth values
55:   zObsBin ← getObsZBin(obsZ)

    ▷ color and surface normal components of the model
56:   if obsNormal is valid then
57:     sigmaBin ← getSigmaBin(expZSigma)    ▷ quantize expected depth uncertainty
58:     probNotHit ← probNotHitGivenDXAndNotMoved[sigmaBin][zExpBin][zObsBin]

59:     if obsRGB is valid then
60:       phit_normalizer ←  $\int_{y=0}^1 \mathcal{N}(y; expRGB.r, \sigma_{rgb}) \times \int_{y=0}^1 \mathcal{N}(y; expRGB.g, \sigma_{rgb}) \times$ 
         $\int_{y=0}^1 \mathcal{N}(y; expRGB.b, \sigma_{rgb})$ 
61:       pcol_hit_pdfval ←  $\frac{\mathcal{N}(obsRGB.r; expRGB.r, \sigma_{rgb}) \times \mathcal{N}(obsRGB.g; expRGB.g, \sigma_{rgb}) \times \mathcal{N}(obsRGB.b; expRGB.b, \sigma_{rgb})}{p_{hit\_normalizer}}$ 
62:       probCM ← 1
63:       probCNM ← p(¬h) × probCM + p(h) × pcol_hit_pdfval
64:     else
65:       probCM ← invalid
66:       probCNM ← invalid

```

▷ move to log space to avoid overflow, which happens for kappas over 1000 or so

```

67:   logPhit_pdfval ← logPhit_pdfval_precalc + κhit × (obsNormal · expectedNormal)
68:   phit_pdfval ← elogPhit_pdfval
69:   probNM ← pnormal_moved_pdfval
70:   probNNM ← p(h) × phit_pdfval + p(¬h) × pnormal_moved_pdfval
71: else

```

```

72:    $\sigma Bin \leftarrow 0$     $\triangleright$  assume largest possible uncertainty
73:    $probCM \leftarrow \text{invalid}$ 
74:    $probCNM \leftarrow \text{invalid}$ 
75:    $probNM \leftarrow \text{invalid}$ 
76:    $probNNM \leftarrow \text{invalid}$ 

 $\triangleright$  depth component: read from tables filled during precomputation
77:    $probXM \leftarrow probDXGivenMoved[\sigma Bin][zObsBin]$ 
78:    $probXNM \leftarrow probDXGivenNotMoved[\sigma Bin][zExpBin][zObsBin]$ 

```

---

## A.2 Approximate Model

An approximation of the above model that doesn't require large tables to be kept in memory. We use the approximation that the distribution of measurements about a surface is Gaussian rather than similar to a Gaussian.

$\triangleright c_d$ , from [59], is the probability of the beam hitting a surface at depth  $l$  given there is a surface there and the beam hasn't hit before  $l$

$\triangleright c_r$ , from [59], is the probability of there being a surface at depth  $l$  along a given beam (different  $l$  values are treated as independent)

$\triangleright p_{bias}$ , from [59], is the fraction of total probability to be allocated to the random-noise part of the model

```

1: function PRECOMPUTE
2:    $expLambda \leftarrow \frac{\log(1-c_r)}{binSize}$     $\triangleright$  match the exponential to the geometric we'd have in the JAIR model
3:    $expNormalizerGivenMoved \leftarrow \int_{y=0}^{maxZ} Exp(maxZ; expLambda)$ 
4:    $expWeight \leftarrow \frac{c_r}{expLambda}$     $\triangleright$  weight the exponential part such that its value at 0 is  $c_r$ 
5:    $numZBins \leftarrow \text{round}(\frac{maxZ}{binSize})$ 
6:    $p\_normal\_moved\_pdfval \leftarrow \frac{p\_normal\_moved\_kappa}{4\pi \times \sinh(p\_normal\_moved\_kappa)}$     $\triangleright \mu = 0$ , so this reduces to constant
7:    $logP\_hit\_pdfval\_precalc \leftarrow \log(p\_normal\_hit\_kappa) - \log(4\pi \times \sinh(p\_normal\_hit\_kappa))$ 

```

---

$\triangleright$  compute  $p(z \mid m = 1)$  and  $p(z \mid m = 0)$

$\triangleright$  args expZ, expZSigma, expRGB, expNormal: expected surface attributes, including depth standard deviation

$\triangleright$  args obsZ, obsRGB, obsNormal: observed surface attributes

$\triangleright$  return: prob{D,C,N}{M,NM}: probabilities of observations in {depth, color, normal} components given that the surface has {moved, not moved}

```

8: function COMPUTE(expZ, expZSigma, obsZ, expRGB, obsRGB, expNormal, obsNormal)

```

$\triangleright$  precompute quantities to be used multiple times

```

9:    $occluderProb \leftarrow \text{if}(obsZ < expZ) \text{then} else(1 - c_d)$ 
10:   $expNormalizer \leftarrow expNormalizerGivenMoved$ 
11:   $normalNormalizer \leftarrow \int_{y=0}^{maxZ} \mathcal{N}(y; expZ, expZSigma)$ 

```

```

12:   $normalPDFVal \leftarrow \mathcal{N}(obsZ; expZ, expZSigma)$ 
13:   $uniformPDFVal \leftarrow \mathcal{U}(obsZ; [0, maxZ])$ 
14:   $expPDFVal \leftarrow \mathcal{E}xp(obsZ; expLambda)$ 

15:   $pHit \leftarrow binSize \times \frac{normalPDFVal}{normalNormalizer}$ 
16:   $pNotHit \leftarrow binSize \times (uniformPDFVal \times p_{bias\_weight} + \frac{expPDFVal}{expNormalizerGivenMoved} \times expWeight \times$ 
     $occluderProb)$ 
17:   $probNotHit \leftarrow \frac{pNotHit}{pHit + pNotHit}$ 

    ▷ color component
18:  if  $obsRGB$  is valid then
    ▷ ensure each distribution is normalized over observed value for constant expected value
19:   $p_{hit\_normalizer} \leftarrow \int_{y=0}^1 \mathcal{N}(y; expRGB.r, \sigma_{rgb}) \times \int_{y=0}^1 \mathcal{N}(y; expRGB.g, \sigma_{rgb}) \times$ 
     $\int_{y=0}^1 \mathcal{N}(y; expRGB.b, \sigma_{rgb})$ 
20:   $p_{col\_hit\_pdfval} \leftarrow \frac{\mathcal{N}(obsRGB.r; expRGB.r, \sigma_{rgb}) \times \mathcal{N}(obsRGB.g; expRGB.g, \sigma_{rgb}) \times \mathcal{N}(obsRGB.b; expRGB.b, \sigma_{rgb})}{p_{hit\_normalizer}}$ 
21:   $probCM \leftarrow 1$ 
22:   $probCNM \leftarrow probNotHit \times probCM + (1 - probNotHit) \times p_{col\_hit\_pdfval}$ 
23:  else
24:   $probCM \leftarrow \text{invalid}$ 
25:   $probCNM \leftarrow \text{invalid}$ 

    ▷ normal component
26:  if  $obsNormal$  is valid then
    ▷ move to log space to avoid overflow, which happens for kappas over 1000 or so
27:   $logP_{hit\_pdfval} \leftarrow logP_{hit\_pdfval\_precalc} + p_{normal\_hit\_kappa} obsNormal \cdot expNormal$ 
28:   $p_{hit\_pdfval} \leftarrow e^{logP_{hit\_pdfval}}$ 
29:   $probNM \leftarrow p_{normal\_moved\_pdfval}$ 
30:   $probNNM \leftarrow (1 - probNotHit) \times p_{hit\_pdfval} + probNotHit \times p_{normal\_moved\_pdfval}$ 
31:  else
32:   $probNM \leftarrow \text{invalid}$ 
33:   $probNNM \leftarrow \text{invalid}$ 

    ▷ depth component
34:   $probXM \leftarrow binSize \times (uniformPDFVal \times p_{bias\_weight} + \frac{expPDFVal}{expNormalizerGivenMoved} \times expWeight)$ 
35:   $probXNM \leftarrow binSize \times (\frac{normalPDFVal}{normalNormalizer} + uniformPDFVal \times p_{bias\_weight} + \frac{expPDFVal}{expNormalizer} \times$ 
     $expWeight \times occluderProb)$ 
36:   $f \leftarrow \frac{obsZSigma}{.003}$  ▷ hack to sort of use observed-depth uncertainty information
37:   $probXM \leftarrow \frac{probXM}{f}$ 
38:   $probXNM \leftarrow \frac{probXNM}{f}$ 

```

---