

Truncated Signed Distance Fields Applied To Robotics

Örebro Studies in Technology 76



Daniel Ricão Canelhas

Truncated Signed Distance Fields Applied To Robotics

Cover image: Daniel Ricão Canelhas

© Daniel Ricão Canelhas, 2017

Title: Truncated Signed Distance Fields Applied To Robotics

Publisher: Örebro University, 2017
www.publications.oru.se

Printer: Örebro University/Repro 09/2017

ISSN 1650-8580
ISBN 978-91-7529-209-0

Abstract

This thesis is concerned with topics related to dense mapping of large scale three-dimensional spaces. In particular, the motivating scenario of this work is one in which a mobile robot with limited computational resources explores an unknown environment using a depth-camera. To this end, low-level topics such as sensor noise, map representation, interpolation, bit-rates, compression are investigated, and their impacts on more complex tasks, such as feature detection and description, camera-tracking, and mapping are evaluated thoroughly. A central idea of this thesis is the use of truncated signed distance fields (TSDF) as a map representation and a comprehensive yet accessible treatise on this subject is the first major contribution of this dissertation. The TSDF is a voxel-based representation of 3D space that enables dense mapping with high surface quality and robustness to sensor noise, making it a good candidate for use in grasping, manipulation and collision avoidance scenarios.

The second main contribution of this thesis deals with the way in which information can be efficiently encoded in TSDF maps. The redundant way in which voxels represent continuous surfaces and empty space is one of the main impediments to applying TSDF representations to large-scale mapping. This thesis proposes two algorithms for enabling large-scale 3D tracking and mapping: a fast on-the-fly compression method based on unsupervised learning, and a parallel algorithm for lifting a sparse scene-graph representation from the dense 3D map.

The third major contribution of this work consists of thorough evaluations of the impacts of low-level choices on higher-level tasks. Examples of these are the relationships between gradient estimation methods and feature detector repeatability, voxel bit-rate, interpolation strategy and compression ratio on camera tracking performance. Each evaluation thus leads to a better understanding of the trade-offs involved, which translate to direct recommendations for future applications, depending on their particular resource constraints.

Keywords: 3D mapping, pose estimation, feature detection, shape description, compression, unsupervised learning

Acknowledgements

Writing a doctoral dissertation is, in spite of the evidence to the contrary (e.g. this document itself and a superabundance of others like it) not an easy task. That is the nature of survivorship bias: we cannot pile up the dissertations that *didn't* make it and measure how tall a stack they make. Under the course of the past few years, I have often felt that my dissertation was close to ending up in the invisible pile of forgotten efforts. So if you, dear reader, happen to be a graduate student reading this and feeling like you are struggling beyond your means, remember: **you are not alone**. I urge you to reach out. Seek help. That being said, I could only have made it this far because of the unwavering support of those around me, and to them I owe an enormous debt of gratitude.

As there are many people and organizations I wish to thank for reasons that range from the intimately personal to the mundanely professional, I prefer to omit the reasons why and simply state their names. I am confident that they will know why, and that is ultimately what matters to me. There are undoubtedly many others whose presence and participation during this period in my life has made the journey easier and much more enjoyable and although I cannot thank them all individually in this small space, I am grateful to them just the same. So, in no particular order but with some distinctions nonetheless, a most sincere

Thank You:

Julia Maier Canelhas Ann-Therese Kirkland Cintia V. Ricão André P. Canelhas
Orebro Universitet The Linux Foundation Monica Ricão Canelhas
Mats Holmberg Robert Lukierski Elin Hylander nVidia Corp Erik Ricão Canelhas
Peter Baustaedter Erik Schaffernicht Imperial College Franziska Klügl Kungliga Tekniska Högskolan
Emanu E. Garnheim Åsa Wilson Anna Rönnbäck
Dimitar N. Dimitrov Susanne Maier Achim J. Lilienthal Sofie Källgren Prime-Sense
The European Union Todor D. Stoyanov Henrik Andreasson Hanme Kim Thomas Zaar
Steven Lovegrove Clare Wadlow Martin Magnusson Sebastiaan Heukels Gregory Koshmak
Seung-Ho Henrik Holmberg Nicholas Honeth Iris Knippels Richard A. Newcombe
Tomasz Kuchner Elin Bergman Andrew J. Davison

Contents

1	Introduction	1
1.1	Background	1
1.2	Contributions	3
1.3	Outline	4
1.4	List of Publications	5
1.5	Symbols and Notation	6
2	Truncated Signed Distance Fields	7
2.1	Distance Fields: Intuition	7
2.2	Truncated Signed Distance Field (TSDF)	10
2.3	Visualization of TSDFs	16
2.3.1	Direct methods	16
2.3.2	Surface Extraction	18
2.4	Interpolation and Gradient Estimation	19
2.4.1	Mathematical Preliminaries	20
2.4.2	Trilinear interpolation	23
2.4.3	Prismatic interpolation	24
2.4.4	Pyramid interpolation	26
2.4.5	Tetrahedral interpolation	27
2.4.6	Nearest Neighbor (winner takes all)	29
2.4.7	Flooring	30
2.5	Gradients	30
2.5.1	Central Differences	30
2.5.2	Forward and Backward Differences	31
2.6	Drawbacks of TSDF Mapping and Work-arounds	31
2.6.1	Memory	31
2.6.2	Sharp Edges	32
2.6.3	Corners in General	33
2.6.4	Surfaces vs. Truncation Distance	33
2.7	Relationship to Occupancy	34

3	Registration of a Depth Image to a TSDF	37
3.1	Representing Motion	38
3.2	Registration	41
3.3	Deriving a Registration Algorithm	45
3.3.1	In relation to ICP	45
3.3.2	In Relation to Lucas-Kanade	47
3.4	Solution	50
3.4.1	Limitations	53
3.5	Results	54
3.6	Discussion	62
3.6.1	Handling Deformations	63
3.6.2	Thoughts on Surface Orientation	65
4	Feature Detection and Description	67
4.1	Noise Filtering of Depth	69
4.1.1	Bilateral Filter	71
4.1.2	Total Variation - L1 Filter	72
4.1.3	TSDF for depth image denoising	72
4.2	Features on Noise-Filtered Depth	72
4.2.1	NARF feature detector	73
4.2.2	NARF feature descriptor	74
4.2.3	Kernel Descriptors	74
4.2.4	Fast Point Feature Histogram Descriptors	74
4.2.5	Evaluation Methodology	75
4.2.6	Feature Detectors	76
4.2.7	Feature Descriptors	77
4.2.8	Results	78
4.2.9	Discussion	81
4.3	3D Feature Detection	83
4.3.1	Harris Corners	84
4.3.2	Derivatives	85
4.3.3	Integral Invariant Features	87
4.3.4	Evaluation Methodology	89
4.3.5	Experimental Results	91
4.4	Discussion	95
5	Compression	97
5.1	Managing memory complexity - Related Work	103
5.1.1	General Purpose Compression	103
5.1.2	Space Partitioning	103
5.1.3	Hashing	104
5.1.4	Moving Volumes	104
5.1.5	Dictionary Learning	104
5.2	Unsupervised learning for TSDF compression	105

5.2.1	Principal Component Analysis (PCA)	105
5.2.2	Artificial Neural Network	106
5.2.3	Methodology	107
5.2.4	Experimental Results	110
5.3	Discussion	116
6	Minimalistic Representations from TSDF	119
6.1	SSSG - Construction	121
6.2	Geometric Place Recognition Using SSSG	126
6.2.1	Related work - 3D-NDT Histogram Matching	127
6.2.2	Random Sample Consensus Matching of SSSG	128
6.2.3	Methodology	131
6.2.4	Experimental Results	133
6.3	Discussion	139
6.3.1	Improvements and Future work	140
7	Conclusion	141
7.1	A Practical Guide to TSDF Mapping	141
7.2	Large-Scale Tracking and Mapping	142
7.3	TSDFs and Shape-based Features	142
7.4	Future Work	143
7.5	Closing Remarks	144
Appendix A	Code listings	147
A.1	C++ code for packing multiple variables into a single byte	147
References		149

List of Figures

2.1	Sensor measurements and distance transform	8
2.2	Mean of distance transforms	8
2.3	Mean of signed distance transforms	9
2.4	Projective signed distance and truncation	11
2.5	Truncated signed distance and weights	12
2.6	Reconstruction using projective TSDF compared to ground truth	14
2.7	TSDF surface convergence given multiple noisy measurements .	15
2.8	Relationship between truncation distance, voxel-size and variance	15
2.9	Visualization of TSDFs using different rendering techniques . . .	17
2.10	Marching Cubes vs. Marching Tetrahedrons - comparison . . .	18
2.11	Linear interpolation in one dimension	20
2.12	Bilinear interpolation	20
2.13	2-simplex interpolation	21
2.14	The interpolation problem	22
2.15	Trilinear Interpolation: surface	23
2.16	Trilinear interpolation: process	24
2.17	Prismatic interpolation: process	25
2.18	Interpolation within a triangular polygon	25
2.19	Tetrahedral interpolation: surface	27
2.20	Splitting of the cube into tetrahedrons	28
2.21	Tetrahedral interpolation: process	29
2.22	Nearest-neighbor “interpolation”: surface	29
2.23	TSDF and sharp corners	32
2.24	Projective TSDF construction at corners	32
2.25	Conversion from TSDF to occupancy	34
3.1	Second-order approximation of sine and cosine functions	40
3.2	TSDF to TSDF registration	44
3.3	Connection between Distance Fields and Voronoi Diagram . . .	45
3.4	Main components of the SDF-Tracker algorithm	52
3.5	Interpolation method vs frame-rate	55

3.6	Surface reconstruction by direct depth-to-TSDF tracking	56
3.7	Absolute Trajectory Error vs. interpolation	57
3.8	Relative Pose Error (translation) vs. interpolation	57
3.9	Relative Pose Error (rotation) vs. interpolation	59
3.10	TSDF embedded in deformation grids	63
3.11	Non-rigid registration example	64
4.1	Side-by-side: RGB and Depth	68
4.2	Depth image gradient magnitudes vs. filtering methods	70
4.3	Power spectrum of a discontinuous signal	71
4.4	NARF feature descriptor	73
4.5	Industrial robot for container unloading	75
4.6	NARF feature detection stabilities	79
4.7	NARF descriptor matching	80
4.8	FPFH descriptor matching	80
4.9	Gradient kernel descriptor matching	81
4.10	Local Binary Patch kernel descriptor matcing	82
4.11	Spin kernel descriptor matching	82
4.12	ShadowArt illustrating silhouette ambiguity	83
4.13	Limits of observability with a projective camera	84
4.14	Volume integral invariant	89
4.15	Signed distance integral invariant	89
4.16	3D Harris features vs. translation	92
4.17	Integral invariant features vs. translation	93
4.18	3D Harris features vs. rotation	94
4.19	Integral invariant sensitivity to truncation	95
5.1	Low bit-rate quantization	98
5.2	Absolute Trajectory Error vs. bit-rate	100
5.3	Relative Pose Error in translation vs. bit-rate	101
5.4	Relative Pose Error in rotation vs. bit-rate	102
5.5	Training data - real-world	108
5.6	Training data - synthetic	109
5.7	Lossy compression resulting in noise reduction	111
5.8	Lossy compression of TSDF - 2D slice	112
5.9	Selective reconstruction of floor surfaces	114
5.10	Large scale mapping using on-the-fly compression	115
6.1	Overlaid surface and scene graph	120
6.2	Harris response function on TSDF model	121
6.3	2D example of Harris feature detection in a TSDF	122
6.4	RANSAC matching of SSSG - candidate hypotheses	129
6.5	SSSG-RANSAC and 3D-NDT-hist. matching heatmaps	134
6.6	Place recognition ROC plot	135

6.7	Precision and Recall SSSG-RANSAC	136
6.8	Precision and Recall 3D-NDT-Histogram	137
6.9	RANSAC translation and rotation errors	138
6.10	SSSG with differentiated edge qualities	139

List of Tables

1.1	Mathematical notation and symbols used in the text	6
3.1	Rigid-body transformation execution timings	41
3.2	Hand-held SLAM Data	58
3.3	Testing & Debugging Data	59
3.4	3D Object Reconstruction Data	60
3.5	Tracking performance comparison	60
3.6	Parameters for tracking performance evaluation	61
4.1	Filter coefficients	87
5.1	Reconstruction and ego-motion estimation errors	111
6.1	Stanford 3D Scenes	132
6.2	Parameters used for the RANSAC place recognition system . . .	133

List of Algorithms

1	Sphere-tracing for ray-surface intersection	16
2	Obtaining 3D filter kernels	87
3	Parallel SSSG construction	126

Chapter 1

Introduction

1.1 Background

Economic forces are already driving robotic solutions to consumer markets. Increasingly, robots are not just manufacturing products, but becoming products themselves. Vacuum cleaning robots have been commercially available for nearly two decades [1] and autonomous lawnmowers are becoming an increasingly commonplace occurrence on the lawns of Swedish home-owners [2]. Even self-driving cars appear poised to make a mainstream breakthrough as a technology, in spite of the additional caution warranted due to the lethal consequences of malfunctions and the need for new legislation to regulate their use.

The faith we place in such potentially dangerous machines is to some extent owed to impressive advances in real-time vision capabilities enabled by more powerful graphics processing units (GPU) becoming increasingly programmable for general purpose applications ¹. Both within and outside the automotive sector, sensor technologies have also been marked by progress. For instance, the Microsoft Kinect camera, released in 2010, inadvertently provided many robotics researchers with an inexpensive depth sensor that featured lower power-consumption, higher resolution and frame-rate compared to many industrial time of flight or light detection and ranging (LiDAR) solutions.

Although the development of autonomous systems appears to progress rapidly, the ingress of general-purpose robots into our homes still seems far off. This may be explained by the fact that mechanical hardware is still expensive to buy. Furthermore, even if the cost of robots themselves may be brought down, what purpose would they serve? If we take the example of an automobile (the manually operated kind), it is a similarly complex and costly mechanical system to buy and maintain, but it is incredibly **useful**, granting freedoms that are still hard to match by other means.

¹The increased programmability of GPUs is enabled by lower-level abstractions from the graphics hardware, provided by frameworks such as CUDA and OpenCL, as opposed to re-purposing the traditional computer graphics pipeline for general computation [3]

What capabilities would a robot need to possess in order to justify our personal investments in them? Should they be able to free us from household chores? Entertain us? Assist us when our bodies no longer allow us to do the things we desire independently? Replace us in the workplace? Augment us? Fight our wars? Regardless of the role envisioned for general purpose robots in our society, there are still enormous challenges in perception, reasoning, and control that will have to be met before reliable, safe and efficient robots can become a reality. There are also social and ethical challenges that undoubtedly arise with the development of increasingly capable robots. How we address those challenges may have profound impacts on our society. Automation is expected to affect the need for human labor in the future [4] and the prospect of large-scale technological unemployment has already prompted a serious discussion regarding the distribution of wealth [5]. Other potential societal impacts include: the nature of warfare [6, 7, 8], the manufacture of goods (including robots themselves) and its subsequent impact on the environment. Robotics and automation are already causing people to stop and ponder fundamental questions such as “What does it mean to be human?” [9] on a much more personal and pragmatic level than previously may have seemed sane.

This thesis is limited to a single topic within a broad field of study related to robot perception. This field encompasses the computational methods that govern how robots create representations of their environments at small, large and very large scales and how they can use these representations to aid them in different tasks. This seemingly innocuous field of study is by no means devoid of social and ethical implications, including possible dual-use [10]. Perusing some of the literature published in conferences on e.g. military technology, we find that topics such as automatic *target* acquisition [11, 12] have a large overlap with methods used for mapping, detecting and recognizing *objects* in robot vision research. Automated mass-surveillance is another direction in which the results presented herein could potentially be applied as e.g. shape-based biometric descriptors that would render texture-based countermeasures, such as *CV-Dazzle* [13] (that avoids automatic face-detection by painting a specific set of contrasting patterns on a person’s face) useless². Either case is an example of applications that are of questionable benefit to humanity when weighed against the potential misuse and erosion of individual privacy and integrity.

The way in which robots “see” is fundamentally different from our own. We do not, as a general rule, build a geometrically accurate mental representation of our environments while constantly keeping in mind the absolute position of our bodies, relative to the maternity clinic at which we were delivered as babies throughout our lifespans. Robots, in a sense, do. At their core, the feature that makes this feat possible for robots is their internal map representation. In this

²In fact, applying additional texture to ones face may actually aid the recovery of shape in some cases.

work, the properties and methods investigated concern one such representation, known as a Truncated Signed Distance Field (TSDF).

This representation stores information at discrete box-shaped locations in space, called voxels (from the words “volumetric” and “pixel”). Voxel-based map representations are not new in robotics, with occupancy grids [14] having been a standard representation for decades. A useful characteristic with voxels is that they create a direct mapping between the memory layout and the map, which makes the retrieval of information about a region and its surroundings trivial, without searching through complex data-structures. Specifically for distance fields, one has the added benefit of pre-computed distances to the nearest surfaces, which has made them useful in applications ranging from collision avoidance to physics simulation. But what about robotics?

The problem this thesis will address is thus: *In the context of a mobile autonomous robot, equipped with dense depth sensors, can a TSDF be used to improve its mapping, perception, and recognition capabilities?*

1.2 Contributions

In the following chapters we will, aided by experiment, study algorithms built around the TSDF representation to attempt an answer to the stated problem. Congruently, I thus claim the following contributions to the field to be of interest to the robotics community:

- A gentle introduction to the TSDF, focusing on surface estimation from depth sensor measurements. The text presented in this thesis puts into context prior works [15, 16] and offers a more pedagogical and pragmatic point of reference.
- A thorough overview of gradient estimation for volumetric images along with benchmarks. Specifically, I show how the gradient estimation relates to the stability of gradient-based features with respect to translation and rotation of the uniform voxel grid.
- An in-depth review of zeroth and first-order voxel interpolation methods and an evaluation of their performance in the context of tracking and mapping applications.
- The derivation of a direct point-to-TSDF pose estimation algorithm from two conceptually different starting points, i.e. Iterative Closest Point, and 3D Scene Flow.
- An evaluation of the noise-filtering capabilities of TSDFs in the context of depth-image feature detectors and descriptors. Scores are provided for detector repeatability and descriptor matching performance, according to practices adapted for visual features.

- Extension of 2D Harris corner detectors to 3D, and evaluation of their performance on TSDF volumes, as a function of different gradient estimation methods.
- Identification of fundamental failure modes of integral invariant feature detectors, when applied to SDF and TSDFs.
- Proposal of novel 3D descriptors, based on PCA and auto-encoder networks.
- Application of these novel 3D descriptors as a means for on-the-fly compression and spatial extension of the mappable volume, with evaluations with respect to tracking performance and mapping quality. Qualitative results for low-level semantic labeling are also provided.
- Proposal of a novel sparse stable scene graph (SSSG) structure that encodes geometric relationships in a TSDF scene model. A high performance GPU-parallel algorithm is also given for efficient extraction of the graph.
- Proposal and evaluation of a novel place recognition system, based on a GPU-accelerated random sample and consensus (RANSAC) matching of SSSGs.

These contributions are found in the relevant chapters, outlined as follows:

1.3 Outline

- **Chapter 2:** The first technical chapter provides a comprehensive introduction to the TSDF as a geometric representation. Here, I explain its mathematical properties and the most commonly used methods for generation, storage, access and visualization. I also discuss some of its flaws and strategies for mitigating them.
- **Chapter 3:** Here, I make the assumption that a moving camera provides us with depth images and derive an algorithm, based on simple least-squares optimization, for how to estimate the 6-axis pose of the camera relative to the TSDF, in real-time. I also provide a range of configurations that allow the algorithm to be scaled down to the performance level of a regular desktop CPU, along with evaluations of the pose estimation on several data-sets.
- **Chapter 4:** In the fourth chapter, I quantitatively analyze the repeatability of feature detectors and matching reliability of image descriptors when computed on depth images that have been filtered through various means, including fusing data into a TSDF from multiple viewpoints. An additional study of feature detectors is done directly in the voxel space, where the sensitivity in feature detector repeatability is presented, conditioned on different gradient estimation methods.

- **Chapter 5:** In the fifth chapter, I present an algorithm for compressing the TSDF that is fast enough to allow for on-the-fly virtual extension of the environment into several orders of magnitude larger spaces with the positive side-effect of rejecting noise and providing low-level semantic labels. The compression method is based on the unsupervised learning of mappings to 3D descriptor-spaces that serve as weak labels of the compressed content.
- **Chapter 6:** Here, an alternative novel light-weight representation is introduced. Discarding the regular grid-based structure of the TSDF, I derive a sparse graph-based representation that explicitly encodes the neighborhood relations between salient points in the geometry. A simple place-recognition system using these graphs is presented and thoroughly evaluated in a simultaneous localization and mapping (SLAM) setting.
- **Chapter 7:** In the seventh and final part of this thesis I offer concluding remarks, summarizing my contributions to the state of the art and identify possible future directions of this line of research.

1.4 List of Publications

The content of this thesis has in part been the subject of previous publications. These are:

- *DR Canelhas*, 2012, “Scene Representation, Registration and Object Detection in a Truncated Signed Distance Function Representation of 3D Space”, Örebro University, Master’s Thesis
(Chapters 2, 3)
- *DR Canelhas, T Stoyanov, AJ Lilienthal*, 2013, “SDF tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images”, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3671-3676
(Chapters 2, 3)
- *DR Canelhas, T Stoyanov, AJ Lilienthal*, 2013, “Improved local shape feature stability through dense model tracking”, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3203-3209
(Chapter 4)
- *DR Canelhas, E Schaffernicht, T Stoyanov, AJ Lilienthal, AJ Davison*, 2017, “Compressed Voxel-Based Mapping Using Unsupervised Learning”, MDPI Robotics 2017, 6(3), 15
(Chapter 5)

- *DR Canelhas, T Stoyanov, AJ Lilienthal*, 2016, “From feature detection in truncated signed distance fields to sparse stable scene graphs”, IEEE Robotics and Automation Letters, Volume 1, Issue 2, pp. 1148-1155

(Chapter 4, 6)

1.5 Symbols and Notation

To aid in the reading of equations, a table of notation and symbols used in this thesis is provided. In the table below, the letters A , a , b and i are used as generic variables. The notation and dimensionality of variables will also be stated in the relevant sections in the text.

Symbol	Description
A	a matrix or set
A^T	transpose of A - the rows of A become the columns of A^T
b	a vector - including one-dimensional vectors, i.e. scalars
\hat{b}	b expressed in homogeneous coordinates i.e. $\hat{b} = \begin{bmatrix} b \\ 1 \end{bmatrix}$
$\ b\ _1$	L1 norm of b - sum of the absolute values of its components
$\ b\ _2$	L2 norm of b - the “length” of b computed as the square root of the sum of its squared components
$ A $	cardinality of A - the number of elements in A . If A is a set , consider the cardinality to be the number of members in the set, e.g. the number of 3D points in a 3D point-set
$\lfloor b \rfloor$	floor of b - b rounded <i>down</i> to the nearest integer
a_i	the i -th element of the set (or matrix) A . Note that $i \leq A $
$\det(A)$	determinant of A
$\text{tr}(A)$	trace of A
$\text{abs}(b)$	absolute value - for a scalar it is equivalent to the L1 norm
$\exp(b)$	exponential - defined, as e^b where e is the irrational number 2.7182818284590... and b is a scalar
$\exp(A)$	matrix exponential - a related concept to the exponential function for scalars, but has a slightly more elaborate definition, see Eq. (3.7)
$\min(b)$	minimum - the smallest value in b
$\min_{\cdot b}(\text{expr.})$	<i>minimize</i> the expression, with respect to b
$i!$	factorial - the product of all integers from 1 to i , inclusive. By definition zero factorial is equal to one, i.e. $0! = 1$
$\sum_{i=0}^{ A }(\text{expr.})$	sum of the values of the expression, as i varies from zero to $ A $. Sometimes this is abbreviated as $\sum_i^{ A }(\text{expr.})$ meaning “sum of the expression over all members of A ”

Table 1.1: Mathematical notation and symbols used in the text

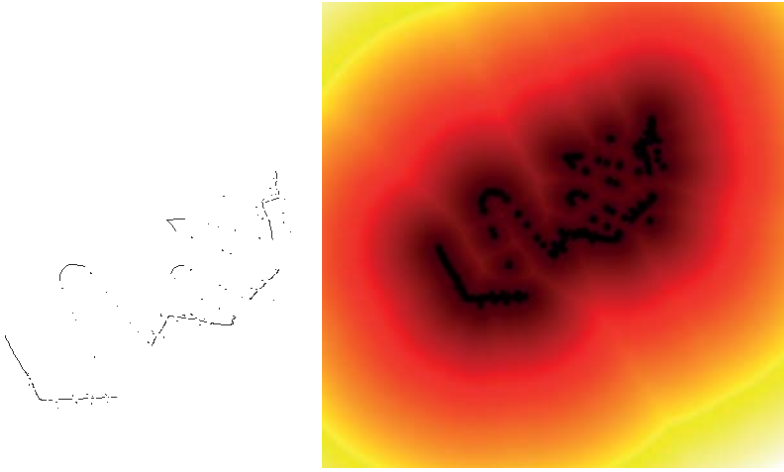
Chapter 2

Truncated Signed Distance Fields

2.1 Distance Fields: Intuition

A distance field is an implicit surface representation. Implicit, in the sense that it describes the space around surfaces, leaving it up to us to infer surface positions and orientations indirectly. Imagine having an exceptionally strong magnet and standing in a room wherein everything was made of iron. The direction and intensity of the pull of the magnet at every location in the room would not technically be a map of the room, but it would allow one to infer very much about the room's geometry. A distance field is a similar abstraction. One that may seem slightly unintuitive, but that makes a lot of sense for a computer program. A distance field is defined as a scalar field whose value at any given point is equal to the distance from the point to the nearest surface.

Robots generally perceive the environment through a series of sensor measurements, so as a visual aid, we will assume a robot equipped with an on-board 2D range-sensor and construct a virtual sensor measurement shown in Fig. 2.1(a). Common artifacts of range-sensor measurements such as noise, occlusions and false surface readings appearing at the edges of geometry have been simulated. For each white pixel in Fig. 2.1(a) we compute the distance from the current (white) pixel to the nearest measurement datum (black) and write this value at the current pixel's position in a new image. In this manner we obtain what is often referred to as the *distance transform* of an image, shown in Fig. 2.1(b), color-coded with brighter color meaning larger distances, for ease of visualization. This distance transform is a discrete, sample-based approximation to the continuous distance field. We will generally deal with discrete fields, sampled on a grid, even though parametric representations are also possible. In practice, interpolation is often used to obtain a continuous estimate of the field.



(a) Synthetic pointcloud in 2D (b) A discretized Euclidean Distance Transform of the point cloud, computed on a regular grid of pixels

Figure 2.1: An illustrative example of a virtual range-sensor's output. The measurements are projected into the 2D space and marked as black dots. In the example, the sensor is assumed to be located in the upper-left corner of the image

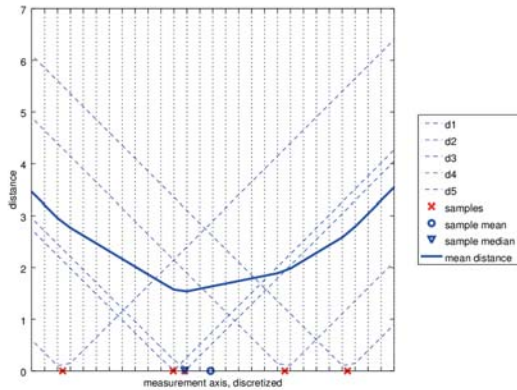


Figure 2.2: Averaging the distance transforms (dotted lines) of randomly distributed samples (red crosses), produces a curve for which the sample mean (blue circle) can no longer be recovered. A median can be obtained by looking for minima in the curve, but this is not necessarily a unique value.

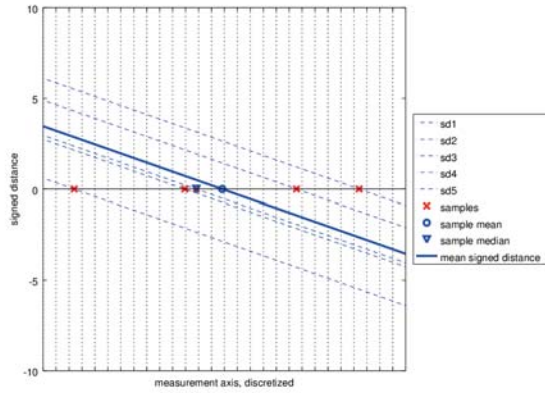


Figure 2.3: The signed distances (dotted lines) to randomly distributed samples (red crosses) can be averaged (thick blue line) with the resulting signed distance transform passing through zero at the sample mean. This zero value can be recovered to good precision by linearly interpolating from the smallest positive and largest negative distance recorded.

Is all the information from our virtual sensor preserved in this field? While the surface position may be approximately recovered by extracting the minimum distance values, we find that the surface orientation is no longer known. In the grid's frame of reference, which side the surface was observed from becomes unclear. Recovering the most likely location of surfaces, given a set of sequential measurements affected by noise is also not a straightforward process as the minimum distance becomes a less distinct value when several distance fields are combined. To illustrate this phenomenon, in Fig.2.2 we see the effect of averaging several **one-dimensional** distance fields, computed based on noisy measurements. The surface measurements are represented as red crosses and the distance field is sampled at regular intervals, marked by vertical dashed lines. The average distance becomes less distinct around the minimum, which coincides with a sample median (which may be non-unique) [17] as it is the solution to

$$\min_x \sum_{k=1}^{|K|} \text{abs}(s_k - x) \quad (2.1)$$

where x is the optimal location of the surface based on $s_k \in K$ one-dimensional measurements of the position.

These two drawbacks are eliminated by using signed distances. The use of a negative sign to indicate distances beyond the measured surfaces causes the average distance to have a zero-crossing that coincides with the sample mean as

shown in Fig. 2.3. Finding the zero-crossing of a linear function is simpler than estimating the minimum of the piecewise linear function that results from the mean of absolute distances in the unsigned case. The positive direction of the gradient at the zero-crossing also reveals how the surface is oriented ¹.

2.2 Truncated Signed Distance Field (TSDF)

Although signing the distance field provides an unambiguous estimate of surface position and normal direction, signed distance fields are not trivial to construct from partial observations except for single objects with nearly complete coverage by a moving sensor [18, 19]. The reason for this difficulty is intuitive: knowing the full shape of an object based only on partial observations is challenging and even seeing the whole object would not reveal its internal structure. Curless and Levoy proposed a volumetric integration method for range images [20] that represents a compromise. It sacrifices a full signed distance field that extends indefinitely away from the surface geometry, but allows for local updates of the field based on partial observations. Their method maintains the properties of signed distances and thus accurately represent surface positions and orientations. This is done by estimating distances along the lines of sight of a range sensor, forming a projective signed distance field, $\hat{D}(x)$.

To explain what is meant by the projective signed distances, let us return to our example range data from Fig.2.1(a). We can compute the line-of-sight distances within the frustum of our sensor using the surface measurements as references for zero (with distances becoming negative for regions behind the surface), as shown in Fig.2.4(a). This is done by assuming each sensor ray to be an instance of the one dimensional case, disregarding adjacent measurements. We call this the projective signed distance field.

Truncating the field at a small negative and positive values, D_{\min} and D_{\max} , respectively, produces the projective truncated signed distance field, shown in Fig.2.4(b). The band wherein the distance field varies between its positive and negative limits, is sometimes referred to as the **non-truncated** region. In other words, a point *outside* the truncated region, is thus located within the narrow band that embeds the surface. Let us label this approximation to the TSDF, based on line-of-sight distances, as $\hat{D}(x)$ and let this be added to the current (n -th) estimate of the TSDF, $D_n(x)$, weighted by a measurement weight $\hat{W}(x)$.

Formally expressing the update rules for the weight and distance value at a given cell location x gives [20]:

$$D_{n+1}(x) = \frac{D_n(x)W_n(x) + \hat{D}(x)\hat{W}(x)}{W_n(x) + \hat{W}(x)}, \quad (2.2)$$

¹As final example, a 3D model embedded in a volumetric signed distance field is made available at <https://github.com/dcanelhas/sdf-dragon>

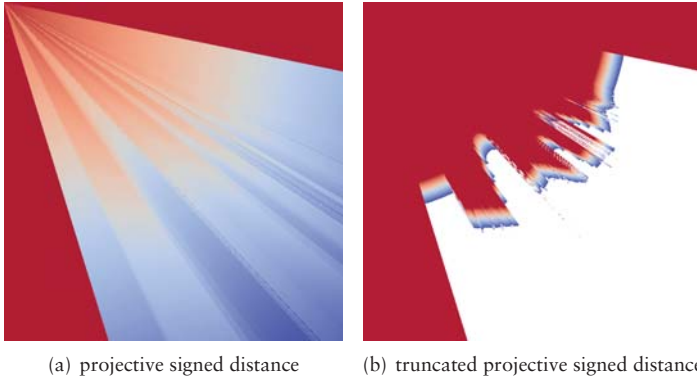
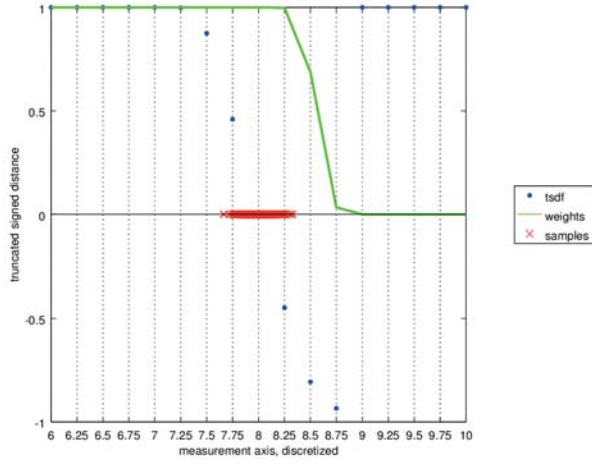


Figure 2.4: projective signed distances, red indicating positive and blue indicating negative distance values, white pixels are uninitialized

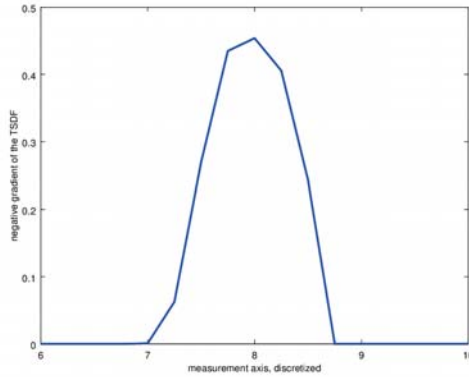
$$W_{n+1}(x) = \min(W_n(x) + \hat{W}(x), W_{\max}), \quad (2.3)$$

where $D_{n+1}(x)$ is thus the updated truncated signed distance at x based on the projective estimate $\hat{D}(x)$. The weight $W_n(x)$ is the accumulated sum (up to a limit W_{\max}) of measurement weights $\hat{W}(x)$. The measurement weight may be an elaborate function modeling the uncertainty in measurement at each updated position x , or in the simplest case of a rolling average, a constant function. Limiting the maximum value of $W_n(x)$ allows for the model to change in order to represent new configurations of the environment and react robustly to dynamic elements. The cell updates can be efficiently done in parallel, since no dependence is assumed between them.

Truncating the distance field is not only practical, but it enables us to represent the noise distribution associated with the measurements from our sensor. To exemplify, we will again look at a one-dimensional case, shown in Fig. 2.5 where the sensor is assumed to be placed on the left side of a surface, and plot the truncated signed distances (normalized to vary between ± 1 outside the truncated region) and weights. For each measurement, updates are done as described in Eq. 2.2 and Eq. 2.3. The resulting (dotted) curve has a sigmoid shape, similar to the error function ($\text{Erf}(x)$) and a similar interpretation is valid, e.g. the true location of the surface has a 50% probability of being between the locations where the TSDF has values of ± 0.5 . Since the error function is the integral of a



(a) Noisy measurements, TSDF and weights



(b) Negative derivative of TSDF

Figure 2.5: The TSDF (blue dots) is computed from the noisy measurements (red markers). The spread of the samples causes the distance field to have a non-linear slope that eases into the truncation limits on each end. The weights are likewise lower on the negative side of the surface (represented by the point where the TSDF passes through zero) due to the distribution of the samples. The negative derivative of the TSDF has some similarities with the distribution that generated the samples.

Gaussian probability density function, its derivative is bell-shaped ², peaking at 8 (also the sample mean).

The line-of-sight distances are Euclidean only in the cases in which the line of sight intersects the surface perpendicularly, in absence of nearby surfaces. It is therefore common to set the weight \hat{W} proportional to the cosine of the angle between the surface normal and the measurement ray [21, 20, 22]. This ensures that the contribution of better (i.e. fronto-parallel) measurements are given higher confidence, but requires an estimate of the surface normal. The update weight can additionally be made dependent on a model of the sensor, attributing lower weight to longer-range measurements, for example. Frisken et al. [23] found that near the surface interface, the distance field can effectively be corrected by scaling it by the local gradient magnitude, though this is not often done in practice to avoid the cost of volumetric gradient computations when updating the TSDF based on new data. We will also opt for the simpler strategy in this work as in practice, the on-board sensor of a robot will move to observe surfaces from a wide range of angles, causing the field to approximate an Euclidean TSDF, as exemplified in Fig. 2.7 and Fig. 2.6. Although reasonably well approximated, deviations are still present at corners evidenced by jitter in gradient orientation.

Since the TSDF is sampled on a regular grid, it is necessary to make a choice about what cell-size to use, and about the width of the non-truncated region around the zero crossings. The cell-size and truncation distance are two inter-dependent variables that may both be selected in relation to the sensor noise variance. In Fig. 2.8, one can see the one-dimensional position error of a reconstructed surface (represented by the zero-crossing of the TSDF) compared to the ground truth surface position. The TSDF is reconstructed based on simulated one-dimensional range-measurements with additive Gaussian noise. The iso-error curves are plotted against cell-size and truncation distances, expressed as a multiple of the variance of the measurements and as a multiple of the cell-size, respectively. The curves reveal a trade-off, where smaller cell-sizes require a wider band around the surface, and vice-versa. There are potentially competing incentives at play, here. It may be desirable to set the truncation at a small value, for instance, in order to avoid interference between the front and back sides of thin objects. One may also desire the highest resolution possible by reducing the cell-size. If both of these attributes are jointly sought we find that, for any isoline of admissible surface position error, the graph represents a Pareto front, since all other choices would be worse in at least one parameter, or infeasible.

As an example, if the variance is estimated to be 0.01m, and allowing for an average normalized (i.e. with the non-truncated region rescaled to ± 1) surface deviation of 0.025, by picking a cell-size of $2.5 \times 0.01\text{m}$, the truncation distance

²Since the sensor was placed on the left side of the surface, the approximation to the error function is reversed, having a negative slope instead of positive. To illustrate the connection to the Gaussian pdf, the derivative has simply been negated, as indicated on the graph.

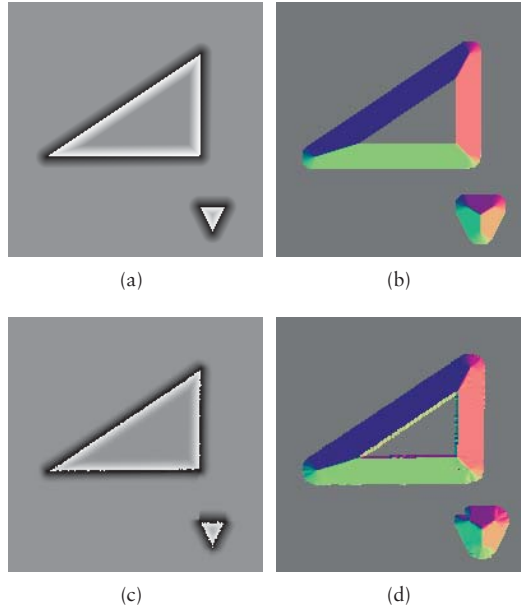


Figure 2.6: In (a) and (b) we see the Euclidean TSDF and its gradient-map, computed from a small environment with two triangular objects. In (c) and (d) we see the TSDF and gradients produced by reconstructing the same scene with measurements generated via a virtual moving depth-sensor.

should be set to no less than $1.1 \times 2.5 \times 0.01\text{m}$, i.e 0.0275m . The given example is marked with a red square on the figure. An additional note about the graph is that since the errors shown in the graph are measured as a fraction of the truncation distance, a point further left on the graph will have a smaller absolute deviation. For sensors that have measurement variance as a function of range, it thus makes sense to increase the truncation distance when updating the TSDF at longer range, since cell-size is usually fixed.

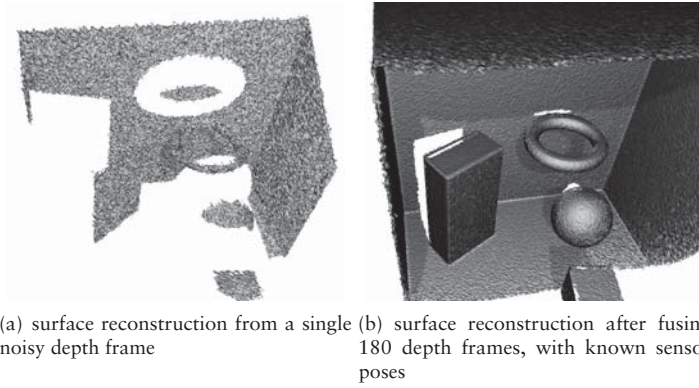


Figure 2.7: Fusing multiple frames with varying viewpoints into a single TSDF allows for noise to be filtered out and fills holes caused by occlusion in the surface reconstruction.

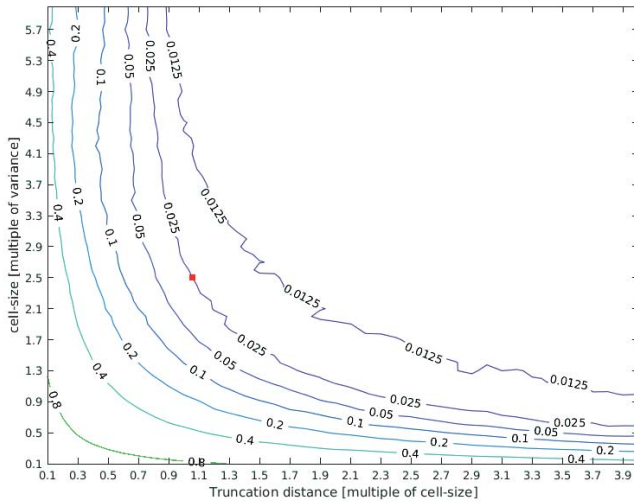


Figure 2.8: The trade-off between cell-size and truncation distance is shown by the error in surface position estimate. Cell-size is measured as a factor to be multiplied by the variance of the measurements, and truncation distance is indicated as a factor to be multiplied with the cell-size. The red marker indicates the example discussed in this section.

2.3 Visualization of TSDFs

In some applications it may be necessary to render the TSDF from a virtual camera, either for visualization purposes or to compare the resulting image with a live frame from a range sensor. The methods related to visualizing implicit surface representations can be divided into methods that directly sample the volumetric representation to produce an image, and those that extract a triangulated surface model that can be rasterized using standard computer graphics pipelines such as OpenGL.

2.3.1 Direct methods

The most common approach to rendering implicit surfaces is by ray-marching, a class of methods that assigns a ray for each pixel of a virtual camera and marches along each ray until an intersection is found. For signed distance fields, the sphere-tracing algorithm [24] sets the step increment along each ray to the distance measured by the field, at the current point along the ray. This allows fast traversal of empty space where distances are large (slightly slower in TSDF, since distance values are limited by truncation). In our discrete representation, as the step increment falls below the size of a voxel, one may continue at constant steps until the first negative distance is obtained and interpolate to find the (zero-valued) intersection based on the last two distance samples. See pseudo-code in Algorithm 1.

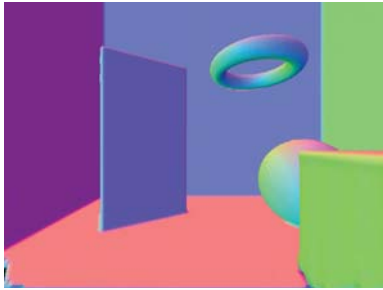
Algorithm 1 Standard depth image ray-marching in a TSDF

```

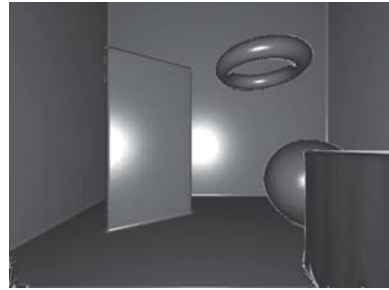
1:  $\alpha \leftarrow 0$ 
2: for  $\forall \mathbf{u} \in I_D$ , over a maximum number of iterations do
3:   compute the ray  $\tilde{\mathbf{r}}$  through  $\mathbf{u}$  originating from  $\mathbf{c}$  using a pinhole camera
     model
4:    $D = \text{TSDF}(\mathbf{c} + \alpha \tilde{\mathbf{r}})$ 
5:   if  $D < 0$  then
6:     interpolate  $\alpha$  based on current and previous  $D$ 
7:     return  $I_D(\mathbf{u}) = \alpha(\tilde{\mathbf{r}}_3)$ 
8:   else
9:      $\alpha = \alpha + D$ 
```

At the surface, one can obtain an estimate for the normal direction by numerically computing the TSDF gradient using a finite differencing method of choice. One can then map the surface normal to a color value directly as in Fig. 2.9(a) or use e.g. Phong lighting [25] as in Fig. 2.9(b).

Depth images can be generated by taking the length of the projection of the intersecting ray onto the camera view axis as the pixel value cf. Fig. 2.9(c). A range image is obtained in a similar fashion by outputting the length of the



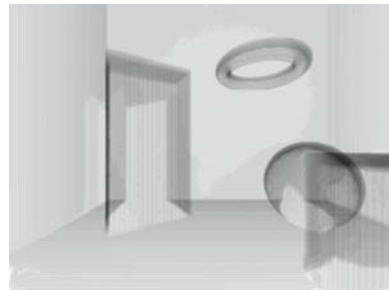
(a) ray-marched intersections colored by the local TSDF gradient orientation



(b) ray-marched intersections colored using Phong lighting

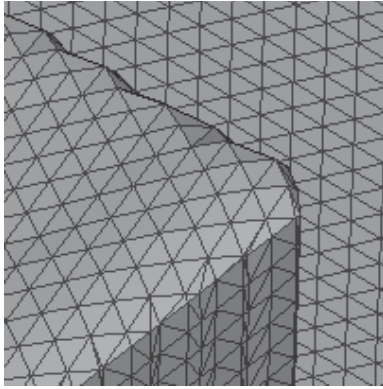


(c) ray-marched intersections with grayscale value proportional to depth (distance along view-axis)

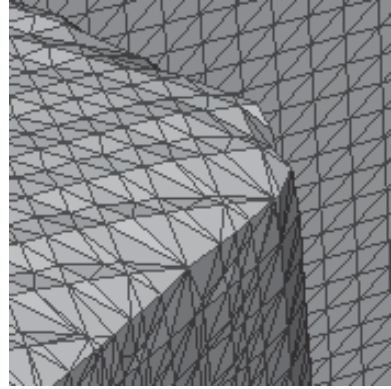


(d) shading based on the number of negative cells intersected as ray traverses the volume

Figure 2.9: A virtual camera/sensor can be used to visualize the volume or generate synthetic measurements, based on the aggregation of real data.



(a) marching cubes extraction of surface



(b) marching tetrahedrons extraction of surface, note the smaller triangle size

Figure 2.10: Visual comparison between marching cubes and marching tetrahedrons for surface extraction

intersecting ray, instead. To obtain a volumetric visualization, one can let the rays traverse the volume up to a predefined distance and count the number of negative voxels intersected cf. Fig. 2.9(d). The volumetric rendering method has the advantage of showing where the reconstruction is incomplete, due to occlusions.

2.3.2 Surface Extraction

Direct methods of visualization may not always be the ideal choice. The TSDF may be supporting several other tasks for the robot and producing visualizations for a human operator may be of lower-priority. If an operator or process requires interaction with the model at a higher frequency than the TSDF can be made available for rendering, a recent view of the model may be sufficient. Sometimes, overlaying different sources of information makes a polygonal surface representation more convenient, and using standard computer graphics for rasterized display is often faster than ray-marching a volume at high frame-rates.

Whereas a virtual camera only sees the information in its field of view, a polygonal surface representation can also be generated for the entire model. This property may be useful for e.g. simulation or planning purposes.

To recover the surface from the scalar field, some polygonization algorithm can be applied. The standard solution to this problem is given by the Marching Cubes [26] or Marching Tetrahedrons [27] algorithms. These algorithms are based on the premise that if one were to construct a cube using 8 neighboring

voxels as vertices, there are only a few possible configurations for a surface passing through it. Classifying the inside and outside status of the neighborhood vertices allows mapping the small region of the scalar field to a set of pre-determined configurations of triangular patches. The exact positions of the vertices that define these triangles can be adjusted to embed them into the zero level-set³, by interpolation. In Fig. 2.10 the same region of a TSDF volume has been extracted using marching cubes and marching tetrahedrons, for comparison. Although Marching Tetrahedrons tends to produce smaller triangles in some cases, there is no compelling reason other than ease of implementation to choose one over the other. Marching Cubes leaves the choice of how to resolve certain ambiguities open whereas Marching Tetrahedrons makes one consistent choice automatically.

2.4 Interpolation and Gradient Estimation

Since the TSDF is stored in a discrete lattice, interpolation is advisable when querying the field at an arbitrary real-valued location. In computer graphics the use of 2D and 3D images as textures applied to continuous surfaces is standard practice, GPUs generally have interpolated access to texture memory built-in for this reason. Bilinear interpolation is the standard operation performed for flat, 2D images, and tri-linear interpolation is its volumetric equivalent.

It is possible to consider potentially more accurate interpolation functions such as cubic or even higher-order polynomials. It may be unwise to do this, however, as we find that the number of samples required to compute the interpolated value increases very quickly. The general formula for the number of samples required for a polynomial fit is given by $s = (o + 1)^n$ meaning that the number of samples (s) needed is at least one more than the order (o) of the polynomial that one wishes to estimate, raised to the power corresponding to the number of spatial dimensions (n). For the trilinear example we get $(1 + 1)^3$ i.e. 8 samples. Tri-cubic interpolation requires 64. It is common to use odd ordered polynomials because they require an even number of samples, which can be taken symmetrically about the query point. Additionally gradient estimation requires sampling the TSDF in at least two locations per dimension to obtain a slope, so the number of memory look-ups needed for computing the gradient is thus at least four (compared to one, for just estimating the value).

Most of the algorithms presented and discussed in this thesis require sampling the TSDF very frequently and when implemented on a GPU, memory access latency is likely to be the bottleneck keeping run-times from decreasing further. This is why our general interest lies not in more complex interpolation schemes (although these may have relevant applications to off-line reconstruction and map-synthesis), but in simpler and faster methods. Nevertheless, we will find, in

³i.e. the set formed by the iso-level of the TSDF with a value of zero

Chapter 4, that more sophisticated methods are sometimes inevitable and return to the topic of gradient estimation with a slightly different perspective.

2.4.1 Mathematical Preliminaries

Consider, as an initial example, the line segment joining nodes a and b in Fig. 2.11. Supposing that we have some arbitrary function ϕ , whose values are known at both a and b . How do we estimate the value of the function at some intermediary point x along the line, i.e. $\phi(x)$?



Figure 2.11: Knowing the relative distances between a , b and x allows interpolating the value at x

One way is to linearly interpolate:

$$\phi(x) \approx \phi(a) \cdot (b - x) + \phi(b) \cdot (x - a) \quad (2.4)$$

In essence, at x , the weight given to the function value at a is proportional to the length of the line segment on the opposite side of the query point i.e., $(b - x)$, and vice-versa. For interpolation in a square, the bilinear approach is simply

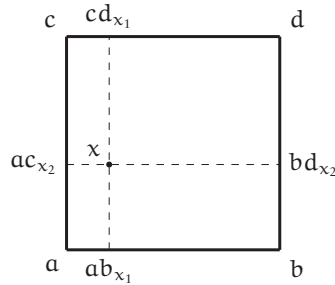


Figure 2.12: Bilinear interpolation offers the choice of linearly interpolating in x_1 first, yielding $\phi(cd_{x_1})$ and $\phi(ab_{x_1})$, and then obtaining $\phi(x)$ or, alternatively starting with x_2 .

an extension of this logic into two dimensions. In other words, for a given query point $x = [x_1, x_2]^T$, shown in Fig. 2.12 to be inside of a square defined by vertices $a, b, c, d \in \mathbb{R}^2$, the function $\phi([x_1, x_2]^T) : \mathbb{R}^2 \rightarrow \mathbb{R}$ can be approximated as:

$$\phi(cd_{x_1}) \approx \phi(c) \cdot (d - cd_{x_1}) + \phi(d) \cdot (cd_{x_1} - c) \quad (2.5)$$

$$\phi(ab_{x_1}) \approx \phi(a) \cdot (b - ab_{x_1}) + \phi(b) \cdot (ab_{x_1} - a) \quad (2.6)$$

$$\phi(x) \approx \phi(ab_{x_1}) \cdot (cd_{x_1} - x) + \phi(cd_{x_1}) \cdot (x - ab_{x_1}). \quad (2.7)$$

Bilinear interpolation scales the contribution of the function value at each vertex by the area of the rectangle formed between the query point and the diagonally opposite vertex. *As a general rule, linear interpolation in any number of dimensions can be performed by scaling the contribution of each vertex by the size of the simplex formed between the query point and the other vertices.* A square is not a simplex in \mathbb{R}^2 , however. A triangle is. If we return to the example shown in Fig.2.12, we see that the point x falls inside both triangles Δadc and Δabc . Denoting the areas of a generic triangle formed by vertices x, y, z as Δ^2xyz we can compactly represent the 2-Simplex interpolated value of $\phi(x)$ as:

$$\phi(x) \approx \phi(a) \cdot (\Delta^2bcx) + \phi(b) \cdot (\Delta^2axc) + \phi(c) \cdot (\Delta^2abx). \quad (2.8)$$

The “size” of any n-dimensional simplex with vertices $v_0, v_1, v_2, \dots, v_n$ can be

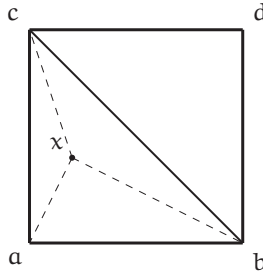


Figure 2.13: 2-simplex interpolation offers the choice of linearly interpolating the value of $\phi(x)$ from one of two triangles. Since x is closest to a , one may prefer the triangle Δabc over Δadc

computed using the following expression [28]:

$$\Delta^n v_0 \dots v_n = \frac{1}{n!} \text{abs}(\det([v_1 - v_0 \quad v_2 - v_0 \quad \dots \quad v_n - v_0])) \quad (2.9)$$

with n indicating the dimensionality i.e. 1 for a line, 2 for a triangle, 3 for a tetrahedron, and so on. By $\det()$ we denote the determinant of the $n \times n$ matrix formed by concatenating the vertex differences as its columns. We now have the basic maths required to understand how the options of simplex interpolation and orthogonal interpolation produce several different ways of estimating the value of a function in between the discrete samples of a voxel grid.

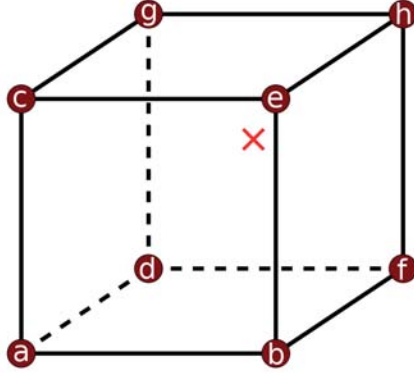


Figure 2.14: The TSDF is stored in a discrete volumetric grid. Interpolation is the process by which we estimate the value of the field at the arbitrary point “X”.

In the following sections we will assume that we have a discrete lattice in 3D, with values sampled at the nodes as shown in Fig. 2.14. The node labels correspond to the following coordinates:

$$\begin{bmatrix} a & b & c & d & e & f & g & h \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \quad (2.10)$$

and we are interested in estimating the value of the underlying function at the real-valued 3D query point x whose components lie in the interval $[0, 1]$.

2.4.2 Trilinear interpolation



Figure 2.15: Ray-marched TSDF with trilinear interpolation used to compute surface intersection and normals

The proverbial workhorse of volumetric interpolation methods is the tri-linear method. It is simple to compute as three consecutive linear interpolations as shown in Fig. 2.16. If we assume the query point illustrated in Fig. 2.14 to have coordinates $\mathbf{x} = [x_1, x_2, x_3]^T$ we can pick an arbitrary dimension along which to interpolate first. Assuming we choose x_1 (as in Fig. 2.16(a)) and using notation consistent with the labels defined in Eq. (2.10) we can express the interpolation for $\phi(\mathbf{x})$ with the steps:

$$\phi(\mathbf{p}) \approx \phi(\mathbf{a}) \cdot (\mathbf{b}_1 - x_1) + \phi(\mathbf{b})x_1 \quad (2.11)$$

$$\phi(\mathbf{q}) \approx \phi(\mathbf{c}) \cdot (\mathbf{e}_1 - x_1) + \phi(\mathbf{e})x_1 \quad (2.12)$$

$$\phi(\mathbf{r}) \approx \phi(\mathbf{d}) \cdot (\mathbf{f}_1 - x_1) + \phi(\mathbf{f})x_1 \quad (2.13)$$

$$\phi(\mathbf{s}) \approx \phi(\mathbf{g}) \cdot (\mathbf{h}_1 - x_1) + \phi(\mathbf{h})x_1 \quad (2.14)$$

abusing the notation somewhat to simultaneously define the interpolated locations as $\mathbf{p}, \mathbf{q}, \mathbf{r}, \mathbf{s}$ and the estimated value of the function $\phi()$ at these locations. This step is followed by,

$$\phi(\mathbf{t}) \approx \phi(\mathbf{p}) \cdot (\mathbf{q}_2 - x_2) + \phi(\mathbf{q})x_2 \quad (2.15)$$

$$\phi(\mathbf{u}) \approx \phi(\mathbf{r}) \cdot (\mathbf{s}_2 - x_2) + \phi(\mathbf{s})x_2 \quad (2.16)$$

defining \mathbf{t}, \mathbf{u} as the interpolated points between \mathbf{p}, \mathbf{q} and \mathbf{r}, \mathbf{s} , respectively. Lastly,

$$\phi(\mathbf{x}) \approx \phi(\mathbf{t}) \cdot (\mathbf{u}_3 - x_3) + \phi(\mathbf{u})x_3 \quad (2.17)$$

Although this process seems to collapse the dimensions of the problem one at a time, down to a point, it is mathematically equivalent to scaling the contribution

of each vertex by the volume of the parallelogram formed between the query point and the vertex diametrically opposed to the vertex in question.

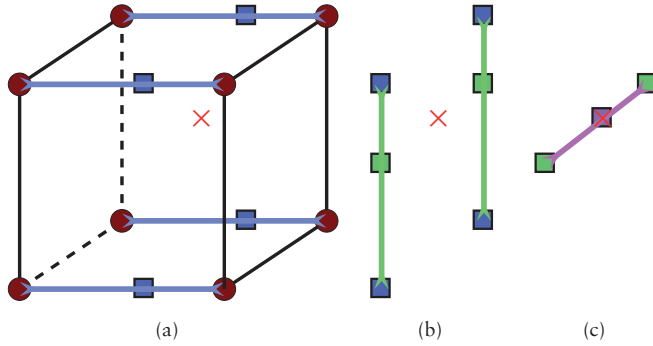


Figure 2.16: Trilinear interpolation

Trilinear interpolation can be thought of in three separate steps as first finding the planar slice within a cube that contains the query point, then by finding a line segment within the slice that contains the point, and finally locating the point along the line. This is mathematically equivalent to weighing the contribution of each vertex in proportion to the volume of the parallelepiped formed between the query point and the diagonally opposite vertex.

2.4.3 Prismatic interpolation

Relative to trilinear interpolation, prismatic interpolation [29] reduces the number of necessary samples to 6 instead of 8. The mathematics involved in computing the interpolated value are also slightly simpler than the trilinear case. One must first assume a direction of “extrusion”. This is the direction along which interpolation will be done last. The choice can be made in advance for the entire map, e.g. parallel with the direction normal to the floor in an indoor environment or dependent on the scaling of the voxels. In Fig. 2.17 we have chosen to extrude along the $[0, 1, 0]^T$ direction.

Regardless of the choice, the first step is determining in which half of the neighborhood cube the query point is located. This test is made using the remaining two dimensions (the ones not considered as the extrusion dimension) which in our example case are x_1 and x_3 . Simply put, if $x_1 + x_3 \geq 1.0$ we should use the vertices associated with the values b, d, e, f, g, h and a, b, c, d, e, g, otherwise (the latter is the case for our example). The inequality stems from the shape of the unit ball defined by the L1 norm.

The next step is to interpolate the function value on each of the triangular faces, e.g using the 2-simplex method detailed in the introduction (or other

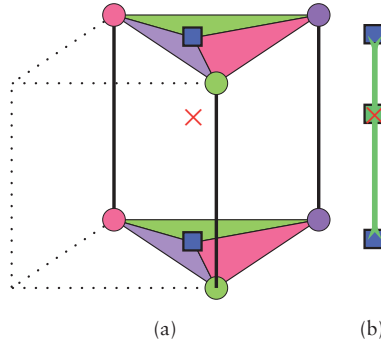


Figure 2.17: Prismatic interpolation

method [30, 31]). The second step is a simple linear interpolation between the resulting values. In Fig. 2.17 we have assumed that 2-simplex interpolation is used and color-coded the vertices and triangles such that the contribution of each vertex is proportional to the area of the triangle with the same color. Since the equations are identical to those described in the Sec. 2.4.1, we will omit the formalism here, however it is worth noting that because the triangular sides of the prism are right-angled triangles with unit catheti, the areas of the subdivisions are straightforward to compute. See Fig. 2.18 for details. The final

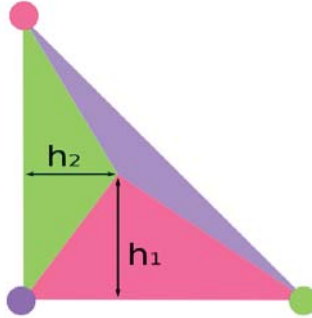


Figure 2.18: The heights of the pink and green triangles are the respective coordinates of the query point, the base is unit. The purple triangle is equal to 0.5 minus the areas of the pink and green triangles

step is a simple linear interpolation in the remaining dimension.

2.4.4 Pyramid interpolation

Removing yet another sample results in the 5-point pyramid interpolation [32] algorithm. Four vertices are chosen on one face of the neighborhood cube and a single vertex is picked on the opposite side. The resulting geometry is a pyramid with a square base and 4 triangular sides. Although the pyramid will in practice need to be oriented different ways to encompass the query point, we will refer to the single vertex that is not on the base as the apex. Options for how to obtain the interpolated value at the query point are several:

- Find the line from the apex that intersects the base while passing through the query point. Perform bilinear interpolation on the square base to get the value at the intersection point, then linearly interpolate between the base-point intersection and the apex for the final value,
- interpolate along each of the four edges connecting the apex to the base, to get a square slice through the pyramid that embeds the query point. Then interpolate on this cutting plane for the final value.
- Interpolate along one dimension of the pyramid base. This results in a line segment across the square base of the pyramid. Form a triangle with this line segment and the apex to get a triangle containing the query point. Interpolate for the final value using any choice of triangular interpolation method.

While this option does result in one less memory look-up than the prismatic method, it leaves many options for the choice of which face to pick as the pyramid base and apex. This choice partly depends on which face is closest to the query point. Given that there are six different faces, and four possible apexes for each face, the number of cases are 24, and there is substantial overlap between them. A possible drawback of this method is that it uses a large proportion of samples from a single side of the cube. This may cause apparent discontinuities in the interpolation when a different configuration becomes necessary as the query point moves within the cube. Implementation-wise this method is slightly more cumbersome, since interpolations are needed in non-orthogonal directions.

2.4.5 Tetrahedral interpolation



Figure 2.19: Ray-marched TSDF with tetrahedral interpolation used to compute surface intersection and normals

When memory access is expensive relative to computation it might be reasonable to look for methods that require as few samples as possible. Tetrahedral interpolation is the method that requires fewest samples [33]. With an additional check, it is also possible to identify the near-degenerate cases, in which a query point is very close to a face, edge or vertex of the tetrahedron. These checks can be done analytically, or by discretizing the sub-voxel space to some arbitrary level of precision to determine how many samples are needed to accurately estimate the value. For example, a query point exactly at the center of the cube could potentially be interpolated by picking any pair of diametrically opposed vertices and performing a single linear interpolation. Checking for near-degeneracy allows for accurate interpolation with even fewer than 4 samples, on average [34].

A cube can be split into a set of tetrahedrons that occupy its entire volume without overlapping in 13 different ways (without counting simple rotations and reflections) [35]. While most splittings result in 6 tetrahedrons, one results in 5, as shown in Fig. 2.20. The 5 part split is obtained by picking any set of four corners such that none of them are joined by a side of the cube, e.g., using the labels defined in Eq. (2.10) we define $T_0 = \{b, c, d, h\}$. These corners form a regular tetrahedron involving the center of the cube. Subtracting it from the cube results in a remaining set of four tetrahedrons ($T_{1..4}$) of identical shape and size. These remaining tetrahedrons are composed by 3 right-angled triangular

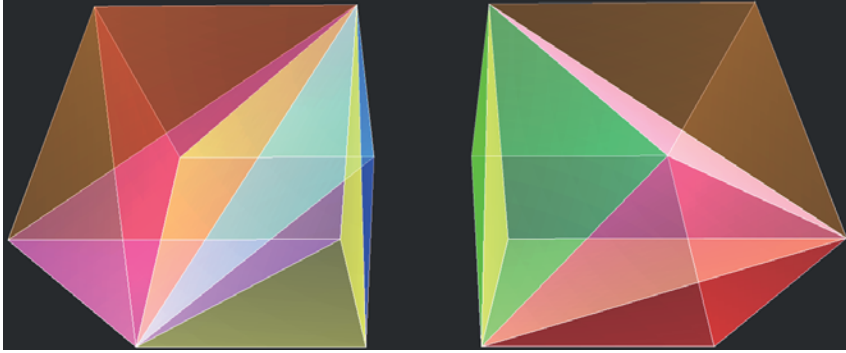


Figure 2.20: Different splittings of a cube into tetrahedrons. On the left side, one of the 12 splittings that produce 6 tetrahedrons and on the right, the splitting that results in 5.

faces and one equilateral triangular face. An example 5-tetrahedron split of the cube uses the following sets of vertices:

$$T_0 = \{b, c, d, h\}, \quad (2.18)$$

$$T_1 = \{a, b, c, d\}, \quad (2.19)$$

$$T_2 = \{e, b, c, h\}, \quad (2.20)$$

$$T_3 = \{g, c, h, d\}, \quad (2.21)$$

$$T_4 = \{f, b, d, h\} \quad (2.22)$$

For interpolation purposes, the motivation for splitting the cubic voxel neighborhoods into 5 tetrahedrons instead of 6 is twofold. Firstly, the resulting shapes have more planes of symmetry (3 and 4) and are closer to isotropic. Secondly, determining which tetrahedron contains the query point can be done more simply with 5 shapes instead of 6.

The interpolation within a tetrahedron (3-simplex) is analogous to the interpolation within a triangle (2-simplex). However, instead of scaling the contribution of the value at a triangle's vertex by the area of the triangle formed with the query point and the remaining vertices, the contribution of the value stored at a vertex in the tetrahedron is scaled by the volume of the tetrahedron formed by the query point and the remaining vertices.

Determining into which tetrahedron a given query point p is found can be done by checking if the distance between the query point and the four corner tetrahedrons is less than unit (by the L_1 -norm). If not, it is found inside T_0 .

The interpolated value for x , when x falls into T_0 is given by:

$$\phi(x) \approx \frac{h\Delta^3 bdcx + b\Delta^3 hcdx + c\Delta^3 hdbx + d\Delta^3 bhcx}{\Delta^3 bdch} \quad (2.23)$$

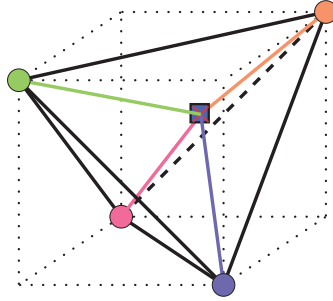


Figure 2.21: Tetrahedral interpolation - the value at the query point marked by “X” is found by taking the sum of the values at each vertex in proportion to the volume of the tetrahedron formed by the remaining vertices and the query point itself. The value is normalized by dividing the result by the volume of the tetrahedron formed by the four vertices

This refers back to Eq. (2.9) for the general formula for the volume of a simplex, however, it is worth noting that the equations for the volumes of $T_{1\dots 4}$ can be greatly simplified, due to their axis-aligned faces.

2.4.6 Nearest Neighbor (winner takes all)



Figure 2.22: Ray-marched TSDF with nearest-neighbor values used to compute surface intersection and normals

The last interpolation method is not much of an interpolation method, at all. Strictly speaking, nearest-neighbor sampling is an extrapolation method. The method is to simply check which octant of the voxel the query point is in and

return the voxel value stored in that octant. It requires a single memory lookup. If we assume I , J and K to be the integer coordinates of the origin of the voxel neighborhood while x , y and z are the fractional parts within it, the following code snippet returns the nearest neighbor in the 3-D array `VoxelGrid`:

```
return VoxelGrid[I + int(floorf(x+0.5))]  
          [J + int(floorf(y+0.5))]  
          [K + int(floorf(z+0.5))];
```

2.4.7 Flooring

Flooring entails simply returning the TSDF as stored at the integer coordinates, completely discarding the fractional part. This is the fastest, but most inaccurate method, with errors potentially as large as the length of a diagonal of a voxel ($\sqrt{3}$ for isotropic scaling, in voxel units). There are few, if any, situations where this would be preferable to any of the other options, especially when considering its negligible performance difference relative to the nearest neighbor method.

2.5 Gradients

Gradient estimation is an important issue for several reasons. The gradient of the TSDF, at the surface interface, relates to the surface normal orientation. Elsewhere, it indicates the direction away from the surface (or towards it, if in the negative region of the field). Local variation in gradients are a measure of *curvature*, which is an important cue for geometric feature detection e.g. for edges, creases and corners. The surface normal is also useful for visualization and for the estimation of properties of the scene that cannot be observed directly [36, 37]. In general, for the applications covered in this thesis, we caution that gradients should be disregarded when a voxel containing a truncated distance value has been used to obtain them.

2.5.1 Central Differences

When computing the gradient of the TSDF, the standard procedure is to use central differences. Central differences are computed by subtracting the TSDF evaluated one voxel in the negative direction from the TSDF evaluated one voxel in the positive direction, relative to the query point. The result is then scaled by $\frac{1}{2}$ (in voxel units) to produce the slope. This is done once for each dimension and results in a vector $\nabla D_n(p) \in \mathbb{R}^3$. Depending on the interpolation method chosen, several grid points will end up being re-used and one will note that the polyhedral memberships (if applicable) and interpolation weights will be the same for each TSDF evaluation, and need not be re-computed.

2.5.2 Forward and Backward Differences

Central differences require 6 TSDF evaluations (2 for each dimension). An alternative, using only 4, is forward (or backward) differences. The latter use one TSDF evaluation where the query point is located and an additional 3, offset in each dimension. The specific application may determine whether it is best to use forward differences, backward differences, to alternate between them or if the resulting estimate is just too inaccurate.

When evaluating the TSDF using the nearest neighbor method, the choice of which points to pick is quite natural; select the nearest voxel from the neighborhood cube and the 3 adjacent corners (connected by an edge). This procedure results in 8 different variations that alternate between forward and backward differences, depending on the location of the query point.

2.6 Drawbacks of TSDF Mapping and Work-arounds

Even though the TSDF offers a straightforward way of representing surfaces with arbitrary topology and provides an efficient mechanism for combining measurements into a consistent surface estimate, there are some cases in which applicability of TSDFs as a surface representation is limited by the hardware or special requirements on surface quality.

2.6.1 Memory

Computationally, operations on a TSDF are generally straightforward to split into independent tasks that may run in parallel. Completing the required operations at a rate comparable to e.g. the frame-rate of a video stream from a sensor in many cases makes the use of a dedicated GPU indispensable. Memory requirements for storing a given TSDF volume scales cubically with the grid resolution (assuming uniform scaling). For applications to mobile robotics, this is unfortunate, since low-latency memory is not abundant on current consumer GPU hardware. However, in a way similar to how an operating system on a computer can extend the available RAM by utilizing non-volatile memory as virtual RAM, a similar reasoning can be applied between the memory available to the GPU and CPU. This is the idea behind what is referred to as *unified memory*. Special care has to be taken to ensure that the use of unified memory does not cause an unacceptable decrease in performance [38]. A system that explicitly manages the transfer of memory between CPU and GPU has been demonstrated to work for reconstructing larger volumes than what would fit in GPU memory at once [39].

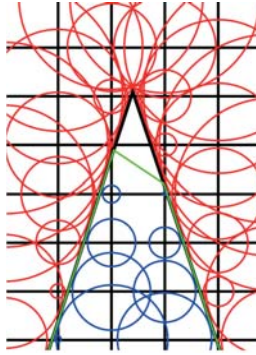


Figure 2.23: A sharp corner embedded in a signed distance field. The lack of negative samples (blue) causes the surface extraction to omit the pointed corner

2.6.2 Sharp Edges

Although the TSDF encodes surface interfaces at subvoxel accuracy through interpolation, sharp corners and edges are not straightforward to extract from a TSDF representation. Surface extraction methods such as marching cubes are simple to parallelize but tend to smooth out sharp features. To gain an intuition as to why this is the case, consider Fig. 2.23. In the figure, each distance sample has been represented as a circle of corresponding radius. Although the intended surface is clearly seen as the red circles overlap, the surface obtained by interpolation results in the green line, missing the sharp point at the top [40]. Computing additional information about gradient orientation as in the Dual Contouring algorithm [41] or feature-preserving surface extraction [42] produces better results at the cost of computing (or storing) gradients in the TSDF.

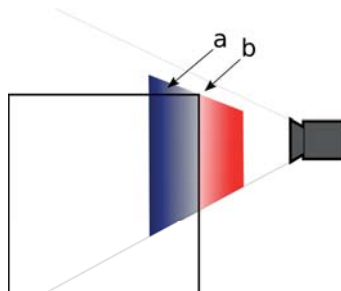


Figure 2.24: Interference between top and side when using the projective TSDF as an input to the reconstruction

2.6.3 Corners in General

Although sharp edges require additional effort to be extracted as a surface from the embedding field, as long as fundamental limits on discrete sampling are not disregarded, the information is still available in the TSDF. However, the issue we are concerned with in this particular section is related to a special case that arises when merging projective distances into the truncated signed distance field. If we look at Fig. 2.24 we note that the region labeled **a** receives a negative update due to the front-facing surfaces. This will cause the region on the top face to bulge outwards slightly. We may also note that due to the assumed independence between rays, the region labeled **b** is being updated towards the upper truncation limit. Even with multiple observations, these literal *corner-cases* tend to remain to some degree. One can mitigate this issue by not performing updates near depth discontinuities in the depth image. Discontinuities can be detected by computing local differences throughout the depth image. Perturbations of the TSDF around corners are quite visible in Fig. 2.6, especially if we observe the variations in gradient orientation. This issue generally highlights the importance of limiting the TSDF to a narrow band around the surface interface. Alternatively one may attempt decrease the weight of TSDF updates in Eq. (2.2) and Eq. (2.3) on the negative side of the field [43]. However, asymmetric weighting strategies risk introducing bias in favor of erroneous measurements in one direction, but not the other, effectively causing surfaces to be reconstructed at an offset from their actual position.

2.6.4 Surfaces vs. Truncation Distance

Truncating the TSDF at a large value ensures that we have a basis for representing the entire noise distribution around the surface, encoding both its mean and uncertainty. However, widening the non-truncated band around the surface also has a detrimental effect on surface fidelity at corners (see Fig. 2.24). Related to this issue, we have the case of thin objects: Either side of an object that is thinner than the width of the negative region of the TSDF will be subject to interference when the other side is being reconstructed using this method. Potential work-arounds such as truncating the TSDF on the negative side earlier, using lower weights for negative distance updates or conditionally updating the cells based on their content tend to distort the estimate of the surface position. These strategies may avoid much worse artifacts such as geometry being left out altogether, though.

When the combination of voxel size and truncation width begin to limit the quality of the surface reconstruction, for a given sensor, there is only so much that can be done by increasing resolution and improving the sensor technology. Ultimately, small-scale details are better captured using a representation that explicitly models surfaces, such as point-based fusion [44, 45] or keyframe based approaches [46, 47], though these trade the coherent spatial organiza-

tion of volumetric representations for increased fidelity and detail in surface representation.

2.7 Relationship to Occupancy

Many published algorithms and software releases in the robotics research community are centered around voxel-based maps that encode *occupancy probability* [14]. This is a concept that deals with which voxels are filled and which are empty, rather than trying to model the shared boundary between them, as TSDFs do. Fortunately, off-the-shelf algorithms designed for occupancy grids can be applied to TSDFs as well; occupancy probability, $f(x)$ can be approximated from a TSDF by a simple affine transformation using the **distance** and **weight**. I propose the following formula:

$$f(x) \approx 0.5 \left(1 - \frac{D(x)W(x)}{D_{\max}W_{\max}} \right) \quad (2.24)$$

D_{\max} and W_{\max} can each be omitted if $D(x)$ and $W(x)$ are in the intervals $[-1, 1]$ and $[0, 1]$, respectively. As an illustration, consider Fig. 2.25, where unseen regions are correctly given a prior occupancy probability of 0.5, the empty regions that have been observed several times have a zero probability of being occupied and the location just behind the surface has a peak which crosses the 0.5 probability boundary at the same location where the TSDF surface crossing occurs. A clamping the values may be necessary to avoid zero

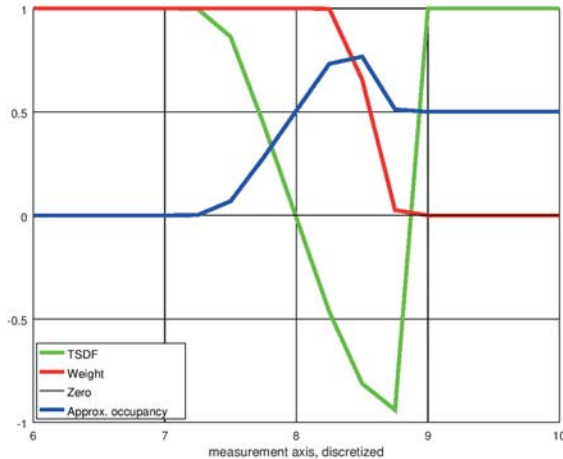


Figure 2.25: A simple affine function can be used to combine weight and truncated signed distance into an occupancy estimate

and unit probabilities, e.g. for updating the map using Bayesian inference. This approximation, using the product between distance and weight, takes into account the number of times a cell has been observed and which state it has been in at all previous measurements. The benefits of varying the truncation distance based on a measure of confidence, as described in section 2.2, also carry over to the approximated occupancy probability, too.

Chapter 3

Registration of a Depth Image to a TSDF

Some of the main features of using TSDFs as maps is that they provide a common frame of reference for storing the aggregated data, while both preserving an implicit encoding of the surface and the variance of the measurements. These properties, together with a simple mechanism for fusing several depth (or range) image frames make them an efficient and useful choice for a map representation.

In the previous chapter we explored some of the numerical properties of the TSDF, as constructed from multiple depth image frames, obtained from known sensor poses. But how would the sensor poses be known? Even in robots equipped with inertial measurement units (IMUs), it is likely that provided poses would only be approximately correct due to drift. Since these devices typically have error-models that are quadratically time-dependent [48] the error in pose estimation tends to accumulate quickly over time. Time-dependent error models also apply for navigation by wheel odometry (or by "step-counting" for legged robots). The problems with assuming that the poses estimated via any of these methods are correct is that they all provide relative estimates that drift, in absolute terms, as time goes by. A way of overcoming this problem is to use the range-sensing devices (and/or cameras) that the robot may have, to find a set of geometric transformations that give a consistent explanation to the observations. This is known as the registration problem.

As one of the first applications of TSDFs to robotics research, we will show here that the TSDF offers an elegant and intuitive way of expressing a pose estimation algorithm, for cases in which depth images are not provided with associated sensor poses (i.e. most cases).

The resulting algorithm is a conceptually simple on-line method for estimating the pose of a depth camera in six degrees of freedom and simultaneously maintaining an updated 3D map, represented as a TSDF, used directly to define a cost function for accurate registration of new data. The proposed algorithm is highly parallel, making it practical for use on modern CPU/GPU hardware,

and achieves good accuracy compared to state of the art methods. It is suitable for reconstructing single household items, workspace environments and small rooms at real-time rates.

The pose estimation problem can be formulated as finding a transformation (or warp) that maps the robot’s current sensory inputs to a frame of reference wherein it *best agrees* with its past observations. This is an intentionally vague statement, to highlight some assumptions that we will make in order to find a solution.

The first assumption is that the transformation is well represented by 3D rigid-body motion of the form $T \in \mathbb{R}^{4 \times 4} \in \mathbb{SE}(3)$ with

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \quad (3.1)$$

where $R \in \mathbb{R}^{3 \times 3} \in \mathbb{SO}(3)$ is a 3D rotation matrix, $t \in \mathbb{R}^3$ is a translation vector and $0^T = [0 \ 0 \ 0]$ is a row-vector of zeros. This representation essentially boils down to stating that all the observed changes are due to the robot’s own motion. By describing the observed changes as a rigid-body transformation, we implicitly assume that the world is static for the duration of the robot’s operation in the environment. While this holds true for most large-scale structures such as buildings, vegetation and furniture, it would be a poor model for a self-driving automobile, whose field of view would often be occupied by other cars. Likewise, people, other robots or animals may be present in the environment and these may need to be actively ignored in the input data or tracked independently for the rigid-body motion model to perform as intended.

We also make an assumption that motion between frames is *small*. This means that a locally optimal strategy for finding the transformation will not converge to an incorrect solution. Small motion is reasonable to assume if the robot is equipped with a sensor that produces measurements at a high frame-rate relative to the robot’s speed (such as a high frame-rate video camera).

So that we do not violate the assumption of small motion, we only estimate the parameters that describe the motion between consecutive frames. This means that each depth image is pre-transformed with the current estimate for the pose and only the incremental change since the last frame needs to be accounted for. Before we go further, we will discuss how we choose to represent 3D rigid-body motion, as there are several possible parametrizations for rotations.

3.1 Representing Motion

Our stated assumptions imply that the transformation will always be close to $T = I$ (a 4×4 identity matrix), as larger rotations and translations would not provide sufficient overlap between consecutive frames to register reliably. In this region it is safe to use a minimal parametrization of rotation, without

risking singularities. Since the motion we are interested in has six degrees of freedom, a minimal parametrization requires six parameters corresponding to three rotations and three translations. Therefore, let $\xi \in \mathfrak{se}(3)$ be a member of the Lie algebra related to the special euclidean group. It contains six elements representing angular and linear velocities, i.e.

$$\xi = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 & v_1 & v_2 & v_3 \end{bmatrix}^T \quad (3.2)$$

Since the Lie algebra is in a tangent-space to the special Euclidean group $\mathbb{SE}(3)$ the parametrization is only valid locally and for accuracy it is still necessary to represent the global motion using a full transformation matrix. This can be done by mapping $\xi \in \mathfrak{se}(3)$ to its corresponding element in $\mathbb{SE}(3)$ and composing the global pose estimate with the additional inter-frame transformation, i.e.

$$T(\xi) = \exp(\Delta t \Xi) \quad (3.3)$$

and

$$T_{n+1} = T(\xi)T_n \quad (3.4)$$

where $\exp()$ is the matrix exponential function, Δt is a duration of time (regarded here as unitary) and $\Xi \in \mathbb{R}^{4 \times 4}$ is the matrix,

$$\Xi = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.5)$$

Where we identify the skew-symmetric matrix containing rotations as,

$$\Omega = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \quad (3.6)$$

The general matrix exponential function is defined as an infinite sum of the form

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!} = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \dots \quad (3.7)$$

which can only be computed approximately. However, due to Ω being a skew-symmetric matrix, this infinite sum can be simplified to a closed-form expression, to directly obtain a rotation matrix from the angular velocity terms. The closed-form expression corresponds to the Rodrigues' rotation formula and requires some algebraic changes to Eq. (3.6). First, we note that the angular velocity components of ξ i.e. ω_1 , ω_2 , and ω_3 are equivalent to an angle-axis representation.

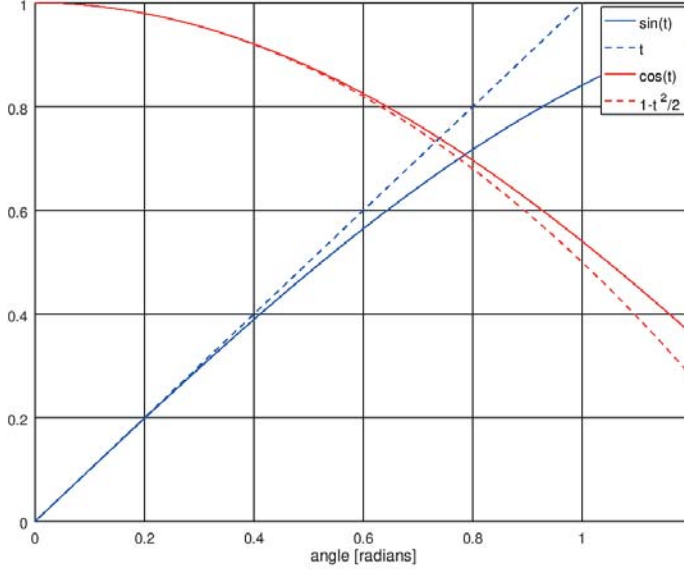


Figure 3.1: sine and cosine functions and their second-order Taylor series approximations around zero. The graph is symmetric for the negative domain

It describes a rotation by an angle of magnitude $\theta = \|\omega\|_2 = \sqrt{\omega_1^2 + \omega_2^2 + \omega_3^2}$ around the unit axis defined by $\mathbf{l} = \frac{\omega}{\|\omega\|_2}$. Separating the angle and axis, and rewriting Ω above we get

$$\theta L = \theta \begin{bmatrix} 0 & -l_3 & l_2 \\ l_3 & 0 & -l_1 \\ -l_2 & l_1 & 0 \end{bmatrix} = \Omega \quad (3.8)$$

and the rotation matrix of Eq. (3.1) is then given by the expression

$$R = I + (\sin \theta)L + (1 - \cos \theta)L^2 \quad (3.9)$$

Our assumption on the motion being small means that we may likewise assume that rotations will be small in magnitude. This allows us to make use of small-angle approximations, i.e. $\sin \theta \approx \theta$ and $\cos \theta \approx 1 - \frac{\theta^2}{2}$. The approximation is much faster to compute, and gives a negligible deviation from the trigonometric functions for small angles cf. Fig. 3.1.

We have discussed how to obtain the transformation from its parametrization as $T(\xi)$ and for completeness, we shall briefly mention how the transformation

can be applied to a given 3D point. The most straightforward way is to write the point in *homogeneous coordinates* which we will denote, for convenience, with a dot ¹ e.g., $\dot{u} = \begin{bmatrix} u \\ 1 \end{bmatrix}$. The point can then be multiplied directly with the 4×4 transformation matrix to apply rotation and translation in a single operation i.e.

$$\dot{p}_b = T_{a \rightarrow b} \dot{p}_a \quad (3.10)$$

However, unless we explicitly *need* the full transformation matrix, e.g. when updating our global pose estimate, there is a more efficient way of transforming a point directly using the parameters contained in ξ . Without explicitly forming the rotation matrix R or the transformation matrix T , one can instead compute,

$$p_b = p_a + \omega_{a \rightarrow b} \times p_a + v_{a \rightarrow b} \quad (3.11)$$

to apply the rotation and translation from coordinate frame a to b .

In Table 3.1 we see the timings relative to obtaining the transformation matrix $T(\xi)$ from ξ as well as transforming a 3D point in MATLAB ² and C++ (using Eigen ³).

	Exponential map	Rodrigues' formula	Rodrigues' formula with small-angle approx.	Matrix-vector multiplication	Cross-product and additions
C++	34.4	19.68	0.786	0.31	0.16
MATLAB	11418	478	321	69	360

Table 3.1: Typical execution times in microseconds (μs) for generating (leftmost columns) and applying (rightmost columns) a rigid-body transformation from its minimal parametrization, using C++ and Matlab

3.2 Registration

As discussed in Chapter 2.3, given a TSDF, we can obtain a depth image by emitting rays from a virtual camera and computing their intersections with the implicit surface boundaries. A bit more formally, we define a depth image as a scalar function $z_n(M)$ that assigns depth to all pixels of each of the $n = 0 \dots N$ frames in a video stream. The domain of z_n is defined on the discretized 2D image plane $M \in \mathbb{N}_0^2$. Formally, $z_n : M \rightarrow \mathbb{R}_+$.

¹Not to be confused with a derivative w.r.t. time. There will be no differential equations here.

²matrix-based programming language and interpreter, <https://www.mathworks.com/>

³C++ template library for linear algebra, <http://eigen.tuxfamily.org/>

Given $z_n(M)$, and knowing the parameters of the depth camera we can also do the inverse, i.e. project the corresponding surface points back into 3D space. Let $s_n(M)$ be the projection of a depth image into 3D. Formally, $s_n : M \times \mathbb{R}_+ \rightarrow \mathbb{R}^3$,

$$s_n(m) = \begin{bmatrix} \frac{m_1 - c_x}{f_x} z_n(m) \\ \frac{m_2 - c_y}{f_y} z_n(m) \\ z_n(m) \end{bmatrix}, \quad (3.12)$$

where $m = (m_1, m_2) \in M$ represents an image pixel and $c_x, c_y, f_x, f_y \in \mathbb{R}$ represent the principal point and focal lengths of a pinhole camera model.

At this point we have at least 3 options for how to proceed with finding the transformation that aligns the depth image to the surface represented by the TSDF volume. We could either:

1. Project the current depth image out into its corresponding 3D point-set, using Eq. (3.2) and register it to a 3D point-set obtained by sampling from the TSDF (i.e. point-to-point);
2. Generate an approximate TSDF volume, using the projective truncated distance transform (see Chapter 2) applied to the current depth image and register the resulting volume to the TSDF volume (i.e. TSDF-to-TSDF);
3. Project the current depth image out into its corresponding 3D point-set, using Eq. (3.2) and register the resulting 3D points to the TSDF volume (i.e. point-to-TSDF).

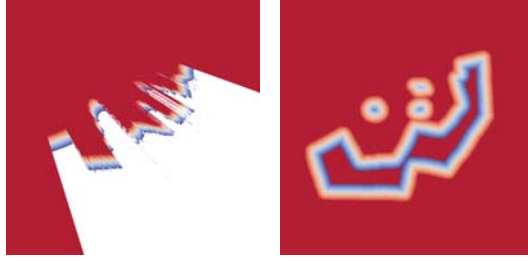
The first option could be achieved in a number of different ways, though a particularly efficient one is to render a depth image of the TSDF from the current estimate of the sensor pose. If this estimated poses have been accurate so far, ray-marching the TSDF produces a denoised depth image with small misalignment relative to the current live frame. The noise characteristics of the rendered depth image tend to improve as the TSDF tracks the mean of the fused surface samples. Registering one point-set onto another, given a rough initial alignment is an extensively researched problem for which the standard solution is to employ some variant of the iterative closest point (ICP) algorithm [49, 50]. In this particular scenario one can do better than the general case, by noting that the surface points of one depth map can be projectively associated⁴ to surface points of the other, as done in *KinectFusion* [21].

The second option is to maintain the model representation as a TSDF and generate a separate TSDF, based on the current live frame and register the two TSDF volumes directly as volumetric images. Standard image alignment methods

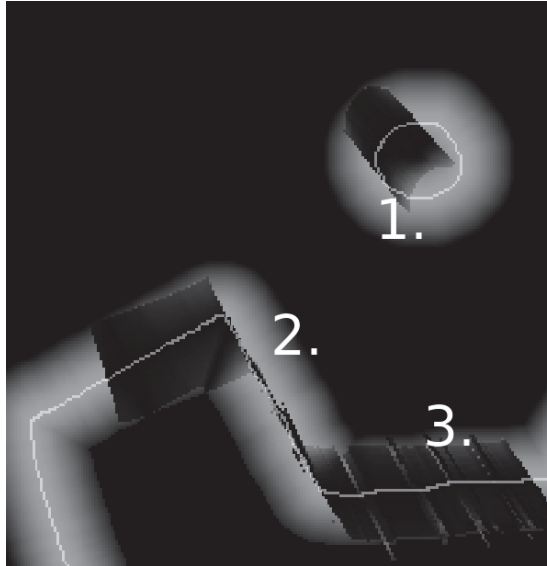
⁴projective association between depth images is achieved by generating the 3D points from the pixels of one image and checking which pixels they project onto in the image plane of the other image

such as the Lucas-Kanade algorithm [51] may then be applied. The described approach is proposed in the *SDF-2-SDF* [52] algorithm. The added computational cost of performing dense voxel-based image alignment is somewhat prohibitive, however. Notably, the *SDF-2-SDF* algorithm proposes frame to frame registration rather than frame to model, needing only to generate a voxel representations for the camera frustums. Even so, the tracking needs to process a large amount of voxels and tends to be much slower than the point-to-point methods. Updating a global map is left as a post-processing step. The frame-to-frame approach of the *SDF-2-SDF* algorithm means that both TSDFs are actually projective TSDFs, and there is no additional information nor denoising gained by extrapolating the depth image into a volume. For any relative motion other than pure rotation, an additional disadvantage is that the projective TSDFs become increasingly misaligned due to the shift in point of origin for the projected distances. One may be tempted to maintain an iteratively updated global TSDF and align a projective TSDF, generated from the current depth image to it, having the benefit of denoising and accurate distances on at least one of the TSDF volumes. This would also represent a questionable approach. As shown in Fig. 3.2, there are several important ways in which the TSDF and projective TSDF differ that may adversely impact the quality of registration.

The third option is our proposed middle-ground, which improves on the convergence properties of the first option and simplifies the process by obviating the need for ray-marching the TSDF to obtain a depth image. Since our proposed method operates directly in the volumetric space, there are no issues with relative scale, as are inherent in projective spaces. Meanwhile, it does not suffer from the same computational complexity as direct volume-to-volume registration, since it operates partly on the depth image coming from the sensor, and thus avoids a fully volumetric framing of the problem. However, it still benefits from having its reference model as an actual TSDF representation in full 3D instead of a view-dependent (and possibly occluded) synthetic camera view of the scene. As our proposed method can be regarded as a bridge between image alignment and point-cloud registration, we will show the derivation of our point-set to TSDF volume algorithm [53, 54] from both perspectives.



(a) Projective TSDF with limited field of view (b) Euclidean TSDF, showing a representation of the whole map



(c) Absolute differences between TSDFs overlaid with surface, cropped

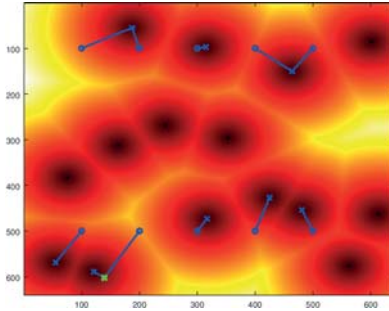
Figure 3.2: Aligning a projective TSDF, generated from a noisy depth image (a) to a “ground truth” TSDF without noise (b) leads to some issues that are not present when aligning a set of points to the TSDF. In the cropped overlay shown in (c), we note that regions 1 and 2 are mismatches whereas 3 is approximately correct near the surface, even though the projective TSDF is affected by some unavoidable noise. The difference at region 1 is due to the projective TSDF not being able to capture the far side of the circular shape that causes changes to the distance field in the interior of the shape. The discrepancy in region 2. is due to the grazing angle at which the distances in the projective case are computed.

3.3 Deriving a Registration Algorithm

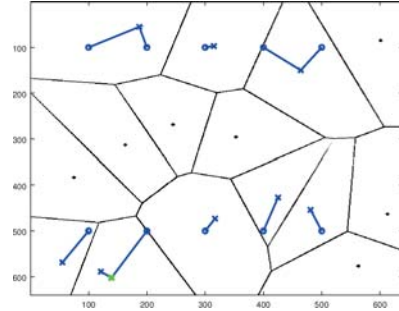
3.3.1 In relation to ICP

The iterative closest point (ICP) algorithm of Besl and McKay [49] is a method for registering 3D shapes and 2D curves. This is done by iteratively finding the rigid-body transformation that minimizes the distances between points on the surface of one shape with the nearest points on the surface of the other shape. ICP monotonically converges to the local minimum, expressed in terms of the distance between the closest points between the respective surfaces. A simple statement of the ICP algorithm can be done in two steps that are repeated in several iterations [55]:

- Step 1 - Correspondence Search. For each point on one surface find an associated point on the other. In standard ICP it is the closest point
- Step 2 - Transformation update. Compute the optimal alignment, given the associated points.



(a) A series of starting points (blue circles) and their convergence towards the closest implicitly defined surface point (blue crosses) by stepping along the negative gradient, the green cross highlights a failure mode, where two iterations were needed due to poor initialization



(b) The derivative of the distance field has its local minima (though not necessarily zeros) wherever the field abruptly changes direction. For signed distance fields this corresponds to the boundaries in the Voronoi diagram.

Figure 3.3: Individual points, descending towards the nearest local minima in the distance field. The figure is described in greater detail in the text.

The error to be minimized is typically of the form,

$$E = \sum_i^{|P|} \min_j \|p_i - q_j\|_2^2 \quad (3.13)$$

stating that the error is a sum of Euclidean distances between the closest point-pairs sampled from two surfaces. Here p_i and q_j are assumed to be points pertaining to the point-sets P and Q of cardinality $|P|$ and $|Q|$ respectively. These point-sets need not contain equal numbers of points, and the closest-point correspondence heuristic often leads to a many-to-one association between the surface points. We will state the pose estimation problem using the error given in Eq. (3.13) and replace p with the surface points obtained from the most recent depth image, by projection into 3D space, i.e., $s_{n+1}(M)$. Finding the optimal 6-DoF⁵ parameters ξ^* is done by solving the minimization problem:

$$\xi^* = \min_{\xi} \sum_i^{|M|} \min_j \|T(\xi)\dot{s}_{n+1}(m_i) - \dot{q}_j\|_2^2 \quad (3.14)$$

It is noted in the original article describing ICP, that the shape representation can be of several types, including “implicit curves: $\bar{g}(x, y, z) = 0$ ” for which a signed distance field is a special case. We note that there are now two nested levels of optimization; an inner level aiming to find the minimum-distance point-pairs and the outer aiming to find the minimum-error transformation given those points. Let $D_n(x)$ be the TSDF, generated from the past n depth images, we thus rewrite the above optimization problem as one with a constraint, as follows:

$$\begin{aligned} \xi^* = \min_{\xi} \sum_i^{|M|} \min_x \|T(\xi)\dot{s}_{n+1}(m_i) - x\|_2^2 \\ \text{s.t. } D_n(x) = 0 \end{aligned} \quad (3.15)$$

For a general implicit curve, the value outside of the surface is not guaranteed to be continuous, bounded or monotonically increasing. For these reasons, obtaining the *closest point* on a general implicit curve, from a query point, may require some local search. The nested optimization problem described by Eq. (3.15) requires a feasible starting point, i.e. a value for x that does not violate the constraint. Among all feasible starting points, we additionally desire the one closest to the transformed point sample $T(\xi)\dot{s}_{n+1}(m_i)$. A reasonable place to start looking for such a point is at the sampled surface point that we wish to compare against. For general implicit surfaces this may not be a straightforward process, but for distance fields it is fortunately quite simple. In Fig. 3.3(a) we see the result of picking a random series of points, computing the negative gradient direction of the distance field and taking steps along it. The step length is equal to the distance field at the initial point. All but one converge to the closest point in a single step. The exception is due to the initialization being on a location where the derivative is poorly defined, and is highlighted in green. This immediate convergence is a consequence of the (signed) distance field being the dual to the Voronoi diagram. Essentially, the minima in the

⁵Six Degrees of freedom: meaning three possible rotations and translations

gradient of the signed distance field correspond to the boundaries of the Voronoi partitions, since this is where the distance field gradient changes direction, cf. Fig 3.3(b). The unsigned distance field additionally has inflection points at the actual surface points and therefore these do not strictly correspond to boundaries in the Voronoi diagram. For sparse points, the signed and unsigned distance fields are obviously equivalent. For continuous surfaces, the partitions in the Voronoi diagram become infinitesimally thin, oriented along the surface normal. Finding the nearest surface point can still be done in the same manner, though.

The relative ease with which we can obtain the nearest surface point within the TSDF means that the constrained inner minimization, i.e. \min_x , can be solved efficiently for x . To that end, we can initialize x at the projected 3D point from the depth image ($T(\xi)\hat{s}_{n+1}$) obtain the closest surface point by taking a step in the negative direction of the normalized TSDF gradient, scaled by the local distance. If the local distance value is positive, this will result in a similar situation as illustrated in the previous paragraph. However, if the distance is negative, the step will be inverted, causing a climb along the gradient back toward the surface. Omitting some subscripts and function arguments for sake of readability, our solution for x can formally be expressed as:

$$x = T\hat{s} - D(T\hat{s})\bar{\nabla}D(T\hat{s}) \quad (3.16)$$

Where $\bar{\nabla}$ indicates a *normalized* gradient. Plugging this definition into Eq. (3.15) lets us express the registration problem as follows (noting that T still depends on ξ , \hat{s} on m_i etc.):

$$\xi^* = \min_{\xi} \sum_i^{|M|} \left\| T\hat{s} - [T\hat{s} - D_n(T\hat{s})\bar{\nabla}D_n(T\hat{s})] \right\|_2^2 \quad (3.17)$$

Since the projected point $T\hat{s}$ appears twice with opposite sign, it results in a zero-vector. The only remaining component that affects the actual norm of the resulting expression is the scaling applied to $\bar{\nabla}D_n(T\hat{s})$ (which itself is of unit length, due to normalization). We note that this is $D_n(T\hat{s})$. Since the objective function is squared (so even negative distances will result in a positive value), we can write as follows (with identical minima):

$$\xi^* = \min_{\xi} \sum_i^{|M|} \left\| D_n(T(\xi)\hat{s}_{n+1}(m_i)) \right\|_2^2 \quad (3.18)$$

Later, we will discuss how to solve this optimization problem to find the optimal camera transformation.

3.3.2 In Relation to Lucas-Kanade

Another way of looking at the problem of registration in TSDFs is to treat both TSDF and the vertices computed from the depth-map as 3D voxel images,

though the second one can be thought of as having non-integer voxel locations. The standard method for image alignment is the Lucas-Kanade algorithm[51], extended to image templates[56].

The original Lucas-Kanade algorithm was formulated as a means to find the translation between two sequential images but can be extended to more general image alignment, including translation, rotation and scale transformations⁶.

Supposing that we have two images $F(x)$ and $G(x)$, we can express the *photometric error* between them as

$$E(h) = \sum_i^{|B|} [F(x_i + h) - G(x_i)]^2 \quad (3.19)$$

for a (voxel) neighborhood B , with h as a parameter by which we can influence the error. To find an estimate for the parameter h , i.e. the displacement of the patch defined by B , we locally approximate the image by its first order Taylor expansion. Recalling the first-order Taylor series approximation of a function:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \quad (3.20)$$

and substituting $x_0 = x$ and $x = x + h$ we get

$$\begin{aligned} F(x + h) &\approx F(x) + \frac{\delta F(x)}{\delta x}^T ((x + h) - x) \\ F(x + h) &\approx F(x) + h^T \frac{\delta F(x)}{\delta x} \end{aligned} \quad (3.21)$$

With this approximation we can compute the derivative of the photometric error w.r.t. the parameter h and equate it to zero to find a minimum, as this is now a simple quadratic function. Setting $\frac{\delta E}{\delta h} = 0$ (assuming partial derivatives organized as a column vector) we get

$$\begin{aligned} \frac{\delta}{\delta h} \sum_i^{|B|} [F(x_i) + h^T \frac{\delta F(x_i)}{\delta x_i} - G(x_i)]^2 &= 0 \\ \sum_i^{|B|} 2 \frac{\delta F(x_i)}{\delta x_i} [F(x_i) + h^T \frac{\delta F(x_i)}{\delta x_i} - G(x_i)] &= 0 \end{aligned} \quad (3.22)$$

and can obtain h .

$$h \approx \left[\sum_x \frac{\delta F(x)}{\delta x} \frac{\delta F(x)}{\delta x}^T \right]^{-1} \left[\sum_x \frac{\delta F(x)}{\delta x} [G(x) - F(x)] \right] \quad (3.23)$$

In Eq. 3.19 the error contribution from a given pixel (or voxel) x is parametrized by a pure displacement h that aims to explain the change in intensity between

⁶e.g. <https://github.com/dcanelhas/sim2-alignment>, 2D image alignment under a similarity transform

images $F(x)$ and $G(x)$. This assumes that the value of the pixel (or voxel) is due to translation (or *flow*), and not brightness changes, and that the spatial gradient $\nabla F(x) = \frac{\delta F(x)}{\delta x}$ exists.

While inserting D_n in the place of $F(x + h)$ in Eq. (3.19) gives us one 3D image, we do not immediately have a corresponding 3D image to plug into $G(x)$.

$$E(h) = \sum_i^{|B|} [D_n(x_i + h) - G(x_i)]^2 \quad (3.24)$$

How about the current depth image? The reason why the depth image is not directly applicable is that depth images are what is commonly referred to as 2.5D, i.e., they have indexable rows and columns, but are not several layers deep, even though each pixel has depth information. However, in a limited sense, depth image *can* be considered equivalent to a volumetric TSDF. We will explore this by way of an analogy. Let us consider the sparse matrix:

$$A \in \mathbb{R}^{10 \times 10} = \begin{bmatrix} 0 & a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c & 0 & d & 0 & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & f & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h & 0 & 0 & 0 \\ 0 & 0 & 0 & i & 0 & 0 & 0 & 0 & j & 0 \\ k & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & l \\ 0 & m & 0 & 0 & n & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Instead of keeping all 100 elements we can use a sparse triplet representation for A , storing row, column and value.

$$A = \{[1, 2, a], [2, 3, b], [3, 5, c], [3, 7, d], [4, 1, e], [5, 6, f], [6, 8, g], \\ [7, 7, h], [8, 4, i], [8, 9, j], [9, 1, k], [9, 10, l], [10, 2, m], [10, 5, n]\}$$

In a similar fashion, for an entry in a fully volumetric 3D image, an equivalent sparse representation would need to store row, column, layer and value as a quadruplet. Now, if all the value components are equal, the value itself can be omitted. If we generalize the concept of indexing to allow for non-integer coordinates, we may define each sparse entry as a vector in \mathbb{R}^3 . The depth image, projected into 3D space i.e., $s_n(M)$, can thus be thought of as a sparse representation of the subset of the TSDF that corresponds to the zero level-set specifically. As there are typically many fewer projected 3D points than voxels it makes sense to perform the summation over the domain of depth image pixels, since those are the only locations where any comparison is meaningful. Noting also that the distance value at the projected 3D point should be identically zero

(due to it representing a surface measurement), we can discard $G(x)$ from the equation. As long as we are querying the TSDF at the locations implied by the projected 3D points, we are implicitly making a comparison with the reference value of zero. This results in an error function dependent only on the squared value of the TSDF at the projected 3D points, translated by h :

$$E(h) = \sum_i^{|M|} [D_n(s_{n+1}(m_i) + h)]^2 \quad (3.25)$$

While the original equation is stated for estimating pixel flow, i.e. pure translation h , to model camera motion, we must make the error dependent on the rigid-body transformation T instead. Obtaining the optimal parameters ξ^* then amounts to solving the following problem, which is equivalent to Eq. (3.18):

$$\xi^* = \min_{\xi} \sum_i^{|M|} \|D_n(T(\xi)\dot{s}_{n+1}(m_i))\|_2^2 \quad (3.26)$$

3.4 Solution

We obtain the solution of the optimization problem defined in either of the two previous sections by linearizing the objective function by means of a first-order Taylor-series approximation around $\xi = 0$, i.e.,

$$\begin{aligned} D_n(T(\xi)\dot{s}_{n+1}(m_i)) &\approx \\ D_n(T(\xi)\dot{s}_{n+1}(m_i))|_{\xi=0} &+ \\ \nabla_{\xi} D_n(T(\xi)\dot{s}_{n+1}(m_i))|_{\xi=0} \xi. \end{aligned} \quad (3.27)$$

The assumption that we are close to $\xi = 0$ holds as long as the inter-frame movement is small. Additionally, as the iterations progress, the parameter vector tends towards zero, making the approximation less inaccurate. Noting that $T = I$ for $\xi = 0$, Eq. (3.27) simplifies to,

$$\begin{aligned} D_n(T(\xi)\dot{s}_{n+1}(m_i)) &\approx \\ D_n(\dot{s}_{n+1}(m_i)) + \nabla_{\xi} D_n(\dot{s}_{n+1}(m_i))^T \xi. \end{aligned} \quad (3.28)$$

In Eq. (3.28) we identify the summand to the Jacobian matrix as,

$$J(m_i) = \nabla_{\xi} D_n(\dot{s}_{n+1}(m_i)), \quad (3.29)$$

where $J(\mathbf{m}_i) \in \mathbb{R}^{6 \times 1}$ can be computed by applying the chain-rule, as follows:

$$J(\mathbf{x}) = \frac{\delta \mathbf{x}}{\delta \xi} \frac{\delta D_n(\mathbf{x})}{\delta \mathbf{x}} \quad (3.30)$$

$$J(\mathbf{x}) = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\delta D_n(\mathbf{x})}{\delta x_1} \\ \frac{\delta D_n(\mathbf{x})}{\delta x_2} \\ \frac{\delta D_n(\mathbf{x})}{\delta x_3} \end{bmatrix}$$

The second part of the Jacobian is obtained by numerical methods, such as finite differences or by convolution with a derivative kernel, as discussed in Chapter 4. We now return to Eq. (3.18) or Eq. (3.26) to plug the new definitions back into the objective.

$$\min_{\xi} \sum_i^{|M|} \left\| D_n(\dot{s}_{n+1}(\mathbf{m}_i)) + J(\mathbf{m}_i)^T \xi \right\|_2^2. \quad (3.31)$$

Expanding the square and simplifying the algebraic expression we end up with,

$$\min_{\xi} \sum_i^{|M|} \xi^T J(\mathbf{m}_i) J(\mathbf{m}_i)^T \xi +$$

$$2\xi^T J(\mathbf{m}_i) D_n(\dot{s}_{n+1}(\mathbf{m}_i)) + D_n(\dot{s}_{n+1}(\mathbf{m}_i))^2. \quad (3.32)$$

Carrying out the sum over the pixels in M for the terms dependent on ξ , we can define the following matrix and vector, respectively,

$$H = \sum_i^{|M|} J(\mathbf{m}_i) J(\mathbf{m}_i)^T, \quad (3.33)$$

$$\mathbf{g} = \sum_i^{|M|} J(\mathbf{m}_i) D_n(\dot{s}_{n+1}(\mathbf{m}_i)). \quad (3.34)$$

Differentiating Eq. (3.32) with respect to ξ and equating the result to zero, we find the least-squares solution ξ^* by,

$$\xi^* = -H^{-1} \mathbf{g}. \quad (3.35)$$

The set of points are transformed by $T(\xi^*)$ and the process is repeated for several iterations. The estimated transform represents the incremental change between

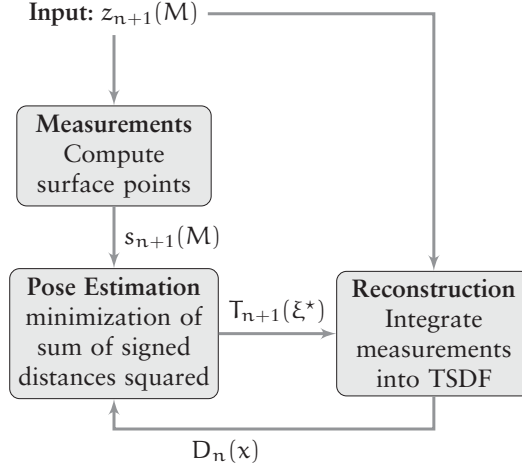


Figure 3.4: Main components of the system, and information flow

the current frame and the previous pose. Since the parametrization of $T(\xi)$ is only valid locally around $\xi = 0$ we reset ξ to zero and pre-transform each newly arrived depth image with a global transformation matrix $\tilde{T}_{n+1} = T(\xi^*)\tilde{T}_n$, with $\tilde{T}_0 = I$, representing the pose change since the beginning of tracking. To summarize, a high-level overview of the system is provided in Fig. 3.4.

Though it is not strictly necessary for convergence, we iterate using a coarse-to-fine sub-sampling on the input depth image, as this provides significant speed-ups in the registration by providing a fast initial alignment [53]. A fixed number of iterations are performed on each level of detail, or until the change in the optimization parameter falls below some pre-defined threshold.

To improve the basin of convergence for the solution, we may scale the contribution of each measurement, based on a weighing function $w(r)$ that takes the residual $r_{n,i} = D_n(\dot{s}_{n+1}(m_i))$ as argument and outputs a scalar weight. The weighing function reduces the influence of outliers on the solution. Changing Eq. (3.33) and Eq. (3.34) to

$$H_w = \sum_i^{|M|} w(r_{n,i}) J(m_i)^T J(m_i), \quad (3.36)$$

$$g_w = \sum_i^{|M|} w(r_{n,i}) J(m_i)^T r_{n,i} \quad (3.37)$$

The weighing function $w(r)$ characterizes an *M-estimator* [57] for which a sensible choice is the Huber estimator [55]. The Huber penalty is dependent on the residual, denoted here by r , and is defined as,

$$w(r) = \begin{cases} 1.0 & \text{if } |r| \leq k \\ \frac{k}{|r|} & \text{otherwise} \end{cases}, \quad (3.38)$$

where k is a small constant. Rescaling with the Huber estimator brings the objective close to an L_1 penalty on the residuals. In practice this absolute value assigns a higher penalty for small residuals compared to a quadratic objective function. This allows for better small-scale adjustments when the image is close to its correct alignment, and is robust to large outliers. However, since we disregard any points outside of the truncation limit for the TSDF, we have implicitly assigned a weight of zero to these anyway. Reducing the influence of measurements associated with objects at a greater distance from the sensor may also be beneficial, as these have a greater uncertainty.

Lastly, we note that adding a small cost related to the norm of the parameter vector ξ itself (an L_2 or Tikhonov-regularizer [58]), has benefits in situations where the solution would otherwise tend to oscillate around the minimum or be poorly constrained by the geometry seen in the environment. Adding $\xi^T \Gamma \xi$ into Eq. (3.29), where $\Gamma \in \mathbb{R}^{6 \times 6}$ is a diagonal matrix attributing cost to the norm of each individual component of ξ , results in,

$$H_{w,r} = \sum_i^{|M|} \Gamma + w(D_n(\dot{s}_{n+1}(m_i))) J(m_i) J(m_i)^T,$$

where $\Gamma = \alpha I$, I being the 6×6 Identity matrix and α a linearly increasing function of the number of the current solution iteration. Large values for α will tend to dampen the amount of change made to the parameter vector at each iteration and such dampening is only interesting once the solution is close to optimal. This type of regularization is the essence of the Levenberg-Marquardt algorithm, applied to ICP as LM-ICP [55]. It can be interpreted as a variable (parametrized by α) trade-off between the Gauss-Newton and gradient descent methods.

3.4.1 Limitations

Since our method uses the TSDF to represent alignment error, it needs a distance function that is truncated at larger values than methods that simply use the TSDF as a scene representation. If a projective TSDF is computed based on the first depth image produced by the sensor, we must ensure that the surface points computed from consecutive depth-image frames find themselves within a region of the TSDF where numerical derivatives can be computed (see Eq. (3.29)) in spite of motion occurring between consecutive video frames. Since the truncation

occurs at larger values, the method is unable to reconstruct details as fine as what would be possible using separate representations for tracking and mapping.

Apart from these limitations, the proposed method may lose track of the pose if the assumption that we are close to the solution does not hold or if the currently viewed geometry does not offer enough variation to constrain all six degrees of freedom, causing the approximate Hessian matrix H in Eq. (3.35) to be close to singular.

3.5 Results

To evaluate the accuracy of the proposed “SDF-Tracker” algorithm in an off-line setting, we tested it on subsets of the RGB-D dataset from the “hand-held SLAM”, “3D Object Reconstruction”, and “Testing & Debugging” categories, provided by the CVPR group at the Technical University of Munich [59]. The data-set consists of several sequences of depth and RGB images, acquired within an office environment. During data-acquisition, the sensor was fitted with reflective markers, enabling an external motion capture system to independently determine drift-free ground-truth poses of the sensor. Sample reconstructions of the scenes, along with plotted camera trajectories can be seen in Tables 3.2, 3.3, and 3.4. Although the data-set contains depth and color images, the latter are not used in any part of this work. The 3D reconstructions shown in the figures are obtained by extracting a triangulated surface from the TSDF, constructed using ground truth poses ⁷.

The parameters and constants introduced in Sec. 3.4 are listed in Table 3.6 with the values used throughout the evaluation. We evaluate the accuracy of our algorithm by computing the absolute trajectory error, relative position error and relative orientation error as indicated in [59]. We compare the results to those of Kinect Fusion (as implemented by the Point-Cloud Library ⁸ [60]) and to two algorithms that use depth information in conjunction with visual features, namely RGB-D SLAM [61], and feature-based 3D Normal Distributions Transform (NDT-F) [62].

As can be seen in Table 3.5, our method achieves similar performance to the tested visual feature-based methods, in spite of tracking from depth alone and not performing any pose-graph optimization. Our method slightly outperforms the reference implementation of *KinectFusion* on the “desk” and “xyz” sequences but fares slightly worse on the “desk2” and “floor” sequences. It is interesting to note that RGB-D SLAM typically has a larger inter-frame error but, due to global optimization and loop-closures, achieves a smaller absolute error. Compared to NDT-F we generally achieve better results. This is expected, since NDT-F

⁷Ground truth poses are only used to generate the visualizations and for evaluating the pose estimates produced by the SDF-Tracker algorithm. They are never provided as an input to the algorithm

⁸ Open Source Kinect Fusion implementation, PCL: <http://svn.pointclouds.org/pcl/trunk>, accessed: Sept. 2012.

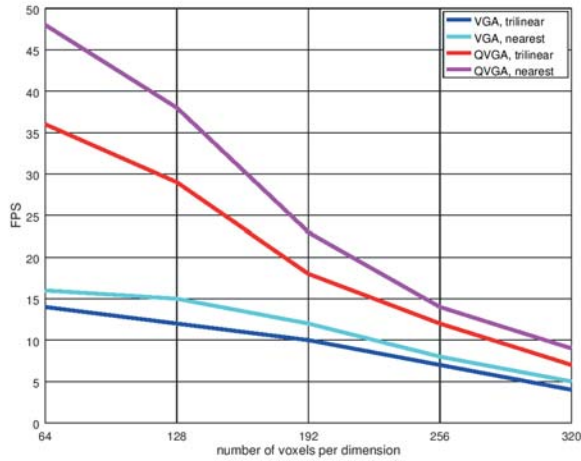


Figure 3.5: Frame-rate plotted as a function of interpolation method, number of voxels and image size.

is a frame-to-frame method and can therefore not take advantage of a map with reduced noise. A notable exception to these statements is the rather poor performance of the proposed algorithm on the “floor” sequence. A closer look at the relative errors in pose estimation for that data set, shown in Fig. 3.6(a) reveals large spikes both in the rotational and translational error, occurring after ca. 25s. Looking at the input data for which this error is produced, we note that it is indeed a representative example for one of the failure modes discussed in Sec. 3.4.1 (namely, H close to singular due to lack of features). The *KinectFusion* algorithm produces similar behavior, for the same reasons.

The run-time of our algorithm, as measured on an Intel Core i7-4770K (3.50 GHz, 4 cores) CPU with 16GB of DDR3 memory at 1600Hz, depends on the number of voxels used and is presented in Fig. 3.5. Performance scales up with the number of cores available on the system and their speed. The algorithm spends the majority of its time evaluating the TSDF. The TSDF needs to be sampled in order to compute the error associated with a surface measurement, but also to compute the derivative of the error relative to position. As a result of this, we find that reducing the complexity of the TSDF interpolation and gradient estimation method results in lower run-time (as seen by the higher frame rate).

But how does the pose estimation suffer in terms of accuracy when using simpler interpolation methods? We can see the performance of the different interpolation methods on a selection of data-sets in Figs. 3.7, 3.8, 3.9. Applying a

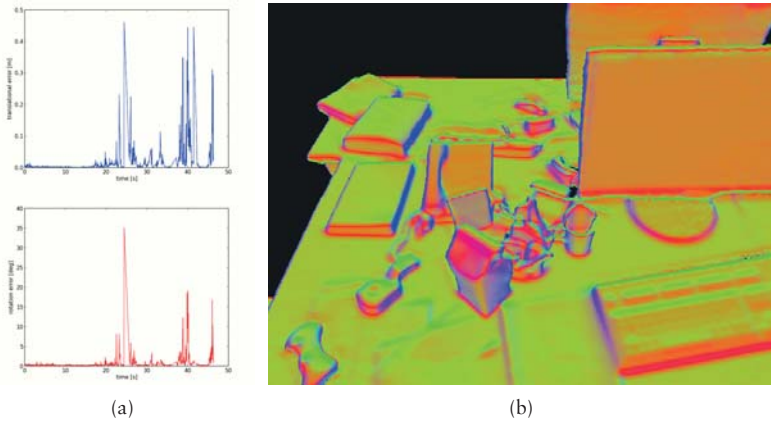


Figure 3.6: Fig. 3.6(a): Relative errors — translation (top) and orientation (bottom). A large spike is seen in both graphs at ca. 25s, caused by an insufficiently constrained solution for the depth-based algorithms. Fig. 3.6(b): Example reconstruction from the fb1_xyz dataset, using 400^3 voxels with 0.006m edge length.

pairwise Mann-Whitney U test [63], we find no statistically significant change in performance between any of them. This means that lowering the computational cost by using a cheaper interpolation method (or none at all in the case of nearest-neighbor) can provide more stable tracking, by allowing higher frame-rates or image resolutions. Which of these attributes to prioritize with the available computational resources is dependent on the lighting conditions and the amount of degradation caused by reducing the shutter time [64] of the camera.

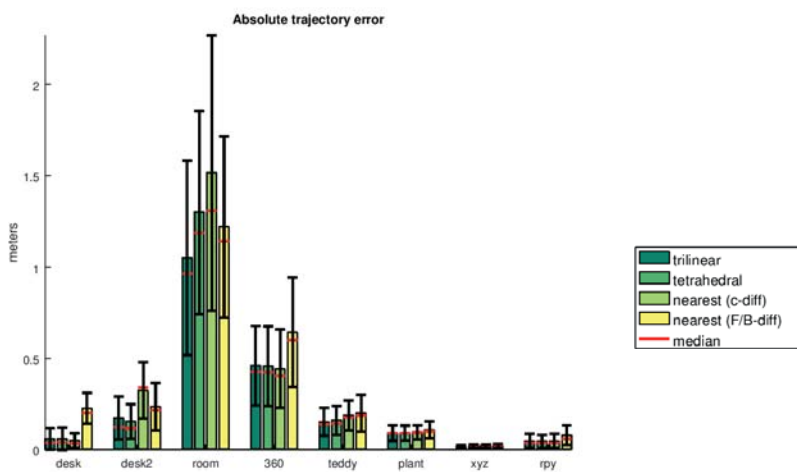


Figure 3.7: Absolute Trajectory Error relative to interpolation method

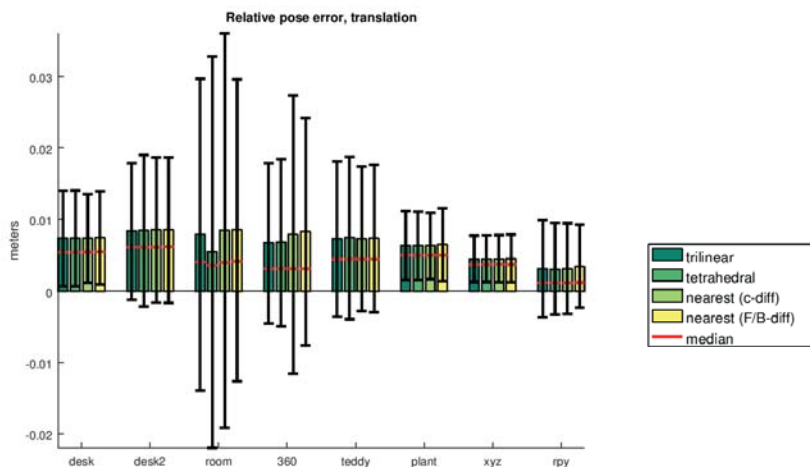


Figure 3.8: Relative Pose Error in translation measured between consecutive frames, relative to interpolation method

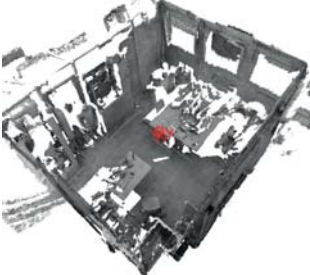

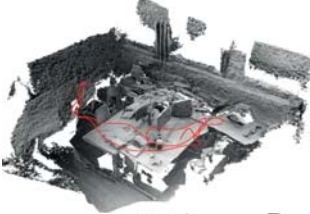
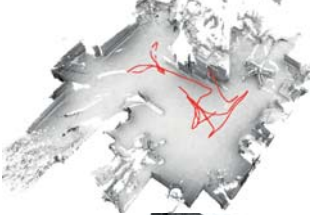
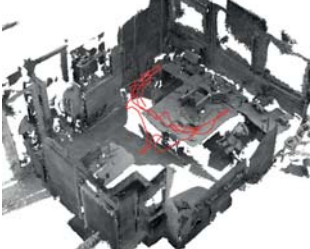
Figure	Name	Trajectory length
	fr1/360	5.818m
	fr1/desk	9.263m
	fr1/desk2	10.161m
	fr1/floor	12.569m
	fr1/room	15.989m

Table 3.2: Selection of sequences from the hand-held SLAM category. 3D reconstruction visualized, with camera trajectory shown in red.



Figure	Name	Trajectory length
	fr1/rpy	1.664m
	fr1/xyz	7.112m

Table 3.3: Selection of sequences from the testing & debugging category. 3D reconstruction visualized, with camera trajectory shown in red.

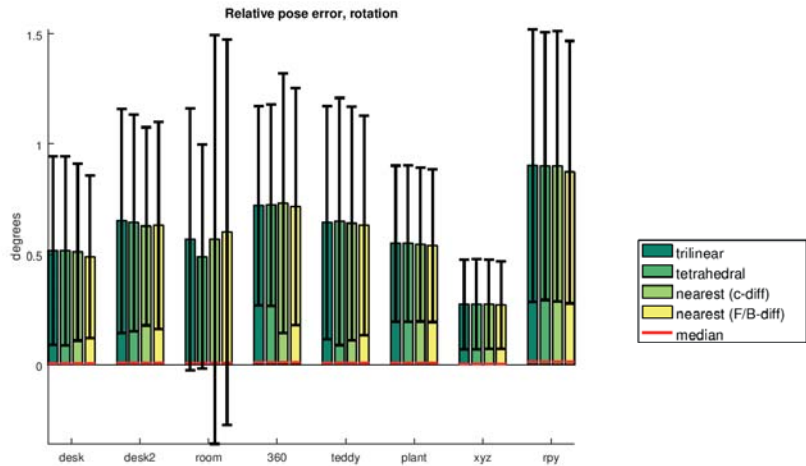


Figure 3.9: Relative Pose Error in rotation measured between consecutive frames, relative to interpolation method

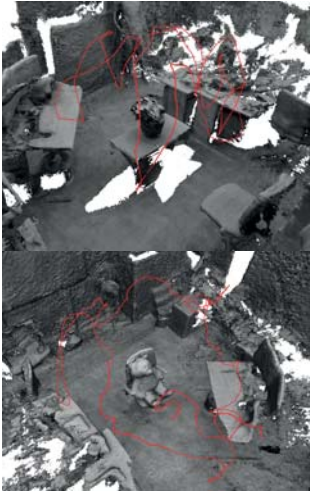
Figure	Name	Trajectory length
	fr1/plant	14.795m
	fr1/teddy	15.709m

Table 3.4: Selection of sequences from the 3D Object reconstruction category. 3D reconstruction visualized, with camera trajectory shown in red

Dataset	Algorithm	Abs err. [m]		Rel err. [m]		Rel err. [deg]	
		RMS	max	RMS	max	RMS	max
xyz (7.11m)	Ours	0.014	0.036	0.003	0.012	0.472	1.810
	PCL-KinFu	0.023	0.070	0.004	0.056	0.474	1.738
	RGB-D SLAM	0.014	0.035	0.006	0.021	0.353	1.633
	NDT-F	0.068	0.125	0.014	0.228	0.844	11.137
desk (9.26m)	Ours	0.033	0.079	0.007	0.051	0.759	3.336
	PCL-KinFu	0.073	0.256	0.020	0.272	2.003	28.317
	RGB-D SLAM	0.026	0.079	0.012	0.063	0.731	6.855
	NDT-F	0.072	0.122	0.019	0.176	1.405	15.358
desk2 (10.16m)	Ours	0.230	0.378	0.019	0.246	1.080	9.785
	PCL-KinFu	0.102	0.312	0.020	0.194	1.795	28.660
	RGB-D SLAM	0.043	0.183	0.018	0.218	1.067	9.632
	NDT-F	0.114	0.249	0.018	0.107	1.219	9.241
floor (12.57m)	Ours	0.984	2.573	0.050	0.462	2.085	35.172
	PCL-KinFu	0.918	1.936	0.035	0.435	1.718	25.175
	RGB-D SLAM	0.035	0.085	0.004	0.027	0.292	1.929
	NDT-F	0.269	0.556	0.025	0.327	0.888	10.728

Table 3.5: Comparative results, This work (Ours), An open-source implementation of the Kinect Fusion algorithm (PCL-KinFu) [21], Feature-based Normal Distributions Transform registration (NDT-F) [62] and RGB-D SLAM [61]

Parameter	value
Voxels	$320 \times 320 \times 320$
Voxel size [m]	0.03
truncation (positive) [m]	0.1
truncation (negative) [m]	-0.06
Huber constant k [m]	0.003
Downsampling levels	3
Downsampling / level	$\times 4, \times 2, \times 1$
Iterations / level	12, 6, 2
Regularization α	$0.001 \times \text{iteration_count}$
Stopping condition (interrupts current level)	$\Delta \ \xi\ < 0.0001$

Table 3.6: Parameters used during evaluation

3.6 Discussion

We have presented a camera-tracking method that uses the 3D scene representation directly as a cost function to perform 6 DoF alignment of 3D surface points. The trajectories estimated by our method, on a well-known dataset, are comparable to those of current state of the art methods, including algorithms which, in addition to depth also employ visual features. Our main contribution lies in a direct model-based approach to on-line registration, as opposed to generating virtual sensor data from a model and performing registration in a sensor-centric frame of reference. This allows for functional camera-tracking and mapping to be done on a system without a GPU.

In our derivation of the objective function from ICP, we started with a point-to-point error metric. However, what we end up with when measuring the distance using the TSDF, is closer to the point-to-plane error. The point-to-plane error considers the projection of the vector connecting two corresponding points onto the normal vector associated with the target point, i.e.,

$$E = \sum_i^{|P|} \min_j \left\| (p_i - q_j)^T n_j \right\|_2^2 \quad (3.39)$$

instead of Eq. 3.13. Close to a surface, where the level-sets of the TSDF tend to be near-parallel, the distance values produced by the point-to-plane error metric and the TSDF error metric are essentially the same. Furthermore, the point-to-plane distance is also negatively signed when the source point (p_i) is behind the surface. Exceptions to the equivalence between the two error metrics are found near corners in the geometry or at isolated points, such as those shown in Fig. 3.3. In such regions, the negative TSDF gradient is oriented towards the surface from every direction, but the normal vector is poorly defined since the planar approximation to a single point is not well constrained enough to be meaningful.

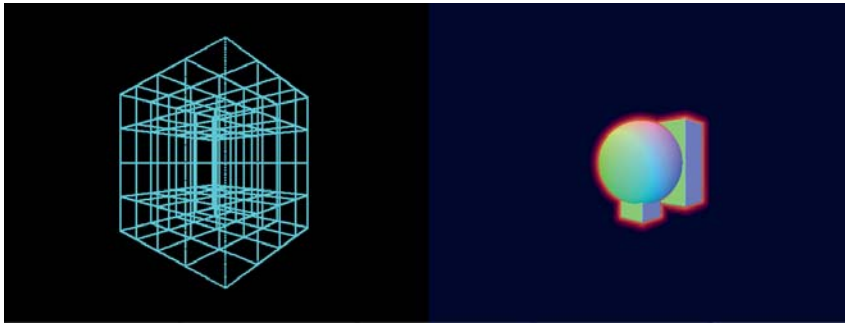
Compared to methods such as *KinectFusion*, which perform a coarse-to-fine ICP registration a depth-image obtained from a sensor relative to a virtual depth image rendered from the TSDF at the current estimated pose, the proposed algorithm requires fewer steps and is simpler to implement. However, it tends to perform equally well in terms of camera tracking, with some minor advantages. It is expected to perform better in regions where the the point-to-plane distance is ill-defined or when alignment between a real and virtual depth image suffers due to occlusions present in the virtual image (which are absent in the full 3D representation).

Although the details regarding interpolation methods and gradient estimation may seem banal, there is evidently some merit in presenting them. A relatively recent evaluation on the state of the art methods for real-time registration of depth images [65] disregarded the algorithm presented here as being computationally infeasible. Their decision to do so rested on the assumption that it

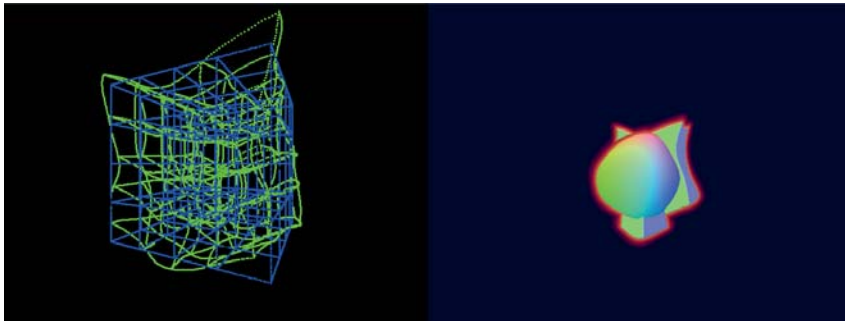
requires both trilinear interpolation and central differences to be viable, when it in fact does rather well without using either.

3.6.1 Handling Deformations

We initially made the assumption that the world is static in order to have a consistent map against which to compare future measurements. This assumption can be relaxed somewhat with non-rigid registration methods. Volumetric image registration is not uncommon in the medical sciences, as many types of patient scans provide volumetric intensity data, with the occurrence of non-rigid warping between scans due to tissue deformation. Embedding the TSDF in a free form deformation grid [66], illustrated in Fig. 3.10 to warp it relative to the input, is a possible approach, but results in a difficult problem due to the abundance of parameters that are unconstrained by the input data. Dealing with the under-



(a) Deformation grid and ray-marched TSDF surface



(b) Randomly displaced deformation grid and ray-marched TSDF surface

Figure 3.10: Embedding a TSDF into a deformation grid allows for warping the space itself. However, the deformations cannot alter the topology since neighboring regions are still connected.

constrained nature of this problem can be done by regularization. One can for example encourage “well-behaved” solutions by initially setting a high, but decreasing penalty on the *curvature* of the deformation grid [67]. This states that translations, rotations, and non-uniform scaling are initially accepted at no cost. As iterations progress, the penalty for causing curvature to the space is reduced and local adaptations are made possible. This form of penalty constrains the parameters to somewhat reasonable-looking solutions, but is slow to converge and tends to spread out the effects of local deformation throughout the deformation lattice. An example of this process is shown in Fig. 3.11 where a sphere is non-rigidly registered to a cube. The main difficulty in using this

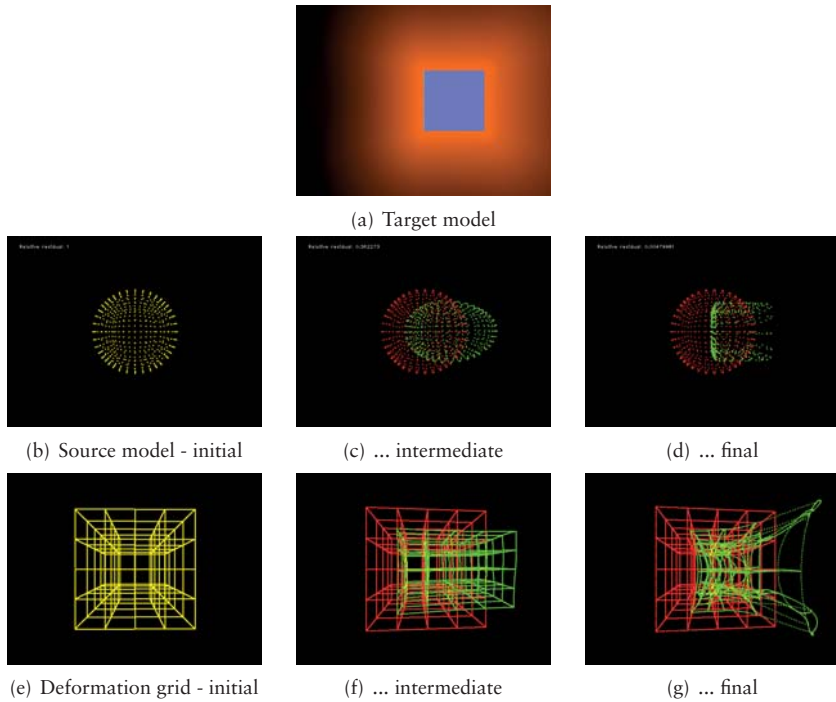


Figure 3.11: Three stages of non-rigidly registering a sphere represented as a point-cloud to a cube, encoded as a TSDF, by updating the positions of the nodes of a deformation grid. Initially the model is translated and stretched into an ellipsoid, then after finding the best alignment, the grid deforms until the points are distributed over the surface of the cube. Note the non-uniform density of the final point distribution in (d) and asymmetry of the grid in (g).

approach for general non-rigid tracking, is that the description does not fit well with how non-rigidity manifests itself in practice. Much of the non-rigid warping

that occurs in the wild is not global in nature, but due to piecewise rigid motion, such as people bending their limbs at the joints, doors opening and closing, and objects changing position within a largely static scene. The sparse and local nature of these deformations can be better described in a more direct way. Efficient real-time approaches to perform template-free non-rigid tracking and mapping have been presented. *DynamicFusion* [68] and *VolumeDeform* [69] are two algorithms that solve the real-time non-rigid tracking and mapping problem from depth and depth plus RGB input, respectively. A detailed overview of these systems is beyond the scope of this thesis.

3.6.2 Thoughts on Surface Orientation

As we discussed in the previous section, the point-to-TSDF error metric has a strong connection to the point-to-plane ICP formulation. An extension to this idea is to utilize information regarding the orientation of surfaces in both the source and the target surfaces. This would, in a sense, be similar to the *SDF-2-SDF* [52] approach mentioned earlier, because it enforces matching of the field behind and in front of the surface, and therefore implies constraints on relative orientations at surface points. Since there can be substantial mismatch between a global TSDF model and a projective TSDF computed from a single depth image, this approach is not ideally suited to frame-to-model tracking but seems to perform well on frame-to-frame alignment. An alternative to aligning TSDF volumes that still enforces the mutual agreement between corresponding surface points and their relative orientations is given by NICP [70]. NICP is an extension to ICP that computes the optimal alignment in a 6-dimensional space of position and surface normal orientation, using a Mahalanobis distance. That is to say, it penalizes distances along the directions of less variance (e.g. out of plane) more than distances within the plane, and allows rotations that maintain surface normals pointing in the same direction.

Chapter 4

Feature Detection and Description

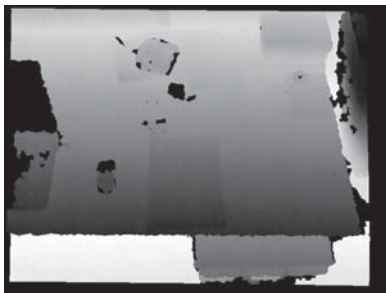
Having the ability to perform tracking and mapping is useful in and of itself, but even if a robot's purpose is merely the autonomous surveying of sites, some form of recognition capabilities is often needed. The necessity stems from the usefulness inherent in realizing when the robot has re-visited the same location again or being able to identify specific objects. The reason why place recognition is important to building maps, is that it provides independent evidence about the robot's location and can be used to produce better estimates about where the robot has been. For autonomous robots fitted with manipulators, recognizing objects is essential to performing more complex tasks such as grasping and manipulating objects.

When performing object recognition, scan alignment or localization, many methods rely on detecting distinguishable regions in the available sensor data. The sensor data at such regions can then be encoded, using a feature descriptor and matched to other descriptors, computed from previous observations or loaded from a database. The idea is that if something "stands out" from the data in one image frame, it is likely to stand out in another. However, matching the data present at these feature locations *directly* is prone to failure, as changes in view-point causes variability in the resulting measurements and the matching process needs to be invariant to these changes. Furthermore, if one has collected hundreds, thousands or even millions of interesting regions, comparing the data directly may be too costly to ensure that a match is recognized in a relevant time-frame for the robot.

For this reason, descriptors are typically devised in such a way that they are of lower dimensionality than the underlying data used to compute them. They may even be ordered in such a way that only the first few elements need to be compared before rejecting a match. Descriptors are often oriented in some consistent frame of reference, such that rotation invariance is obtained. While

this process is well understood for local visual features, algorithms that operate on depth data are a relatively recent development that has received less attention.

Depth data is of high relevance in robotics — virtually all autonomous mobile robots are equipped with a range sensing device. It is thus not surprising that shape-based feature descriptors have been proposed and used in robotic applications such as scan alignment [71], place recognition [72, 73, 74], and object detection [75, 76], to name but a few. Many of the recent contributions focus on improving the consistency of salient feature detectors, improving descriptors to become robust to viewpoint variations and noise as well as finding suitable metrics for matching them.



(a) Depth image (contrast enhanced for visibility)



(b) RGB image

Figure 4.1: Side-to-side comparison between a depth image and RGB color image, both obtained using an Asus XtionPRO live sensor

Although the world has plenty of visible texture, geometrically it tends to be quite smooth on the macroscopic scale. Fig. 4.1 shows a side by side comparison of a typical office desk with some moderate clutter. There is notably less variation in the depth image, than in the RGB domain. This is an illustration of the differences between depth and RGB images, and how precious few the geometric saliences tend to be. Even in natural environments, where geometric variation tends to be more plentiful, it is typically still orders of magnitude sparser than discernible texture variation. This is because differences in the intensity of reflected light will usually be influenced by shape but also texture and illumination [77]. As a consequence, the light intensity variation in an image can be thought of as a strict superset of that caused by shape alone.

Because of the practical relevance of depth images in robotics and their sparse nature, we feel compelled to contribute to their reliability, and in this chapter will discuss ways in which we can use a TSDF to do just that.

In this chapter we will:

- compare the performance feature detectors and descriptors, computed on depth images, subject to denoising the depth image data by either fusing several depth images into a TSDF and synthesizing a denoised version or by applying single-frame filtering algorithms;
- compare the performance of different feature detectors, applied directly in the volumetric TSDF domain. One type of feature detector included in the evaluation is based on gradient orientations. We will therefore study its stability with respect to different gradient estimation methods. Two other types of detectors included are based on integration within a volumetric boundary. For these, we will identify both practical and theoretical limitations on their applicability to our problem domain.

4.1 Noise Filtering of Depth

Filtering the depth sensor noise is not in general guaranteed to result in better feature detection or feature description results, since removing the noise may come at the price of smoothing out important geometric structure. We examine two standard noise removal approaches — bilateral filtering [78] and total variation L_1 norm (TV-L1) filters [79], and compare their effect on feature stability and accuracy to that of TSDF denoising. In order to evaluate the effect of these noise removal approaches, we will make use of several recent depth-based feature detection and description algorithms.

In order to be useful in a place recognition or object detection context, feature detectors and descriptors should be robust to camera viewpoint changes. Therefore, we will evaluate the stability of the Normal Aligned Radial Feature [80] (NARF) interest point detector over increasing baselines in translation and rotation. We shall also compare the success in matching of NARF, Fast Point Feature Histogram (FPFH) [71] and Depth Kernel [75] descriptors, over the same range of motion.



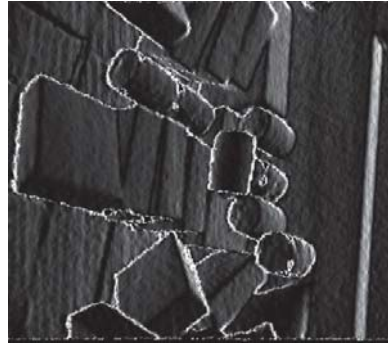
(a) Raw depth image obtained from the sensor



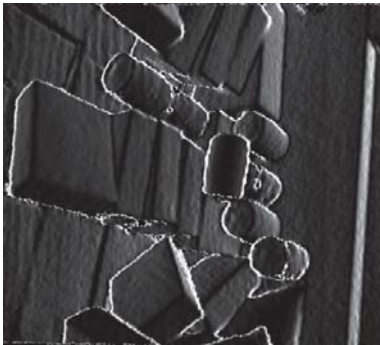
(b) Bilateral filter



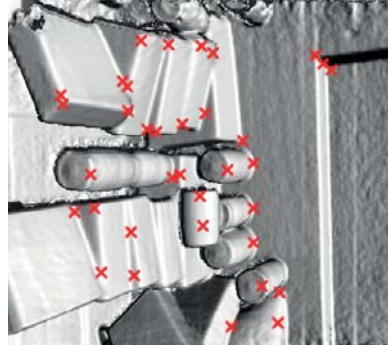
(c) TV-L1 filter



(d) Incremental TSDF denoising (past measurements only)

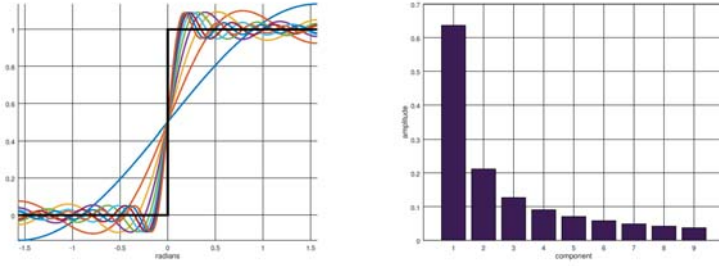


(e) Full TSDF denoising (all measurements)



(f) Reconstructed TSDF surfaces along with manually selected feature point locations

Figure 4.2: Gradient magnitude for typical depth images, processed with different noise filtering techniques



(a) Fourier series approximation to a square wave (b) Amplitude per harmonic component

Figure 4.3: Sharp discontinuities are high-frequency signals and therefore are affected severely by low-pass filters, such as the Gaussian blur filter

From a signal processing point of view, filtering is the act of decreasing the amplitude of unwanted frequencies in the signal. We already know that the world is expected to be geometrically smooth, and therefore we expect that depth sensor measurements should in general be a low-frequency signal. However, to represent discontinuities, such as those seen at the boundaries of objects, we need to maintain high frequency content. The reason for this can be understood by looking at the Fourier series approximation of a square wave (see Fig. 4.3) and noting that the amplitude of higher frequency components are non-negligible (the amplitude of each component is inversely proportional to its respective frequency, and the total sum of amplitudes is unbounded). Fortunately, there are some edge-preserving filters that we can apply. In this section I outline relevant noise filtering approaches for depth image denoising. Examples of a raw depth image, as well as the filtered outputs produced by the discussed approaches are shown in Fig. 4.2 and further explained in the next subsections.

4.1.1 Bilateral Filter

The bilateral filter is a nonlinear filter that updates a pixel p in an image I with a weighted sum of the values of its neighbors q . Unlike a regular Gaussian smoothing filter, which takes into account only the distance between pixels p and q in the image plane, the bilateral filter also considers the difference in *intensity* between these pixels. Formally [81],

$$\text{BF}[I]_p = \frac{1}{W_p} \sum_{q \in S} N_{\sigma_s}(\|p - q\|_2) N_{\sigma_r}(\|I_p - I_q\|_2) I_q \quad (4.1)$$

where N are Gaussian probability density functions (PDFs) with variances σ_s in pixel-space and σ_r in intensity-space, and W_p is a normalization factor. The

bilateral filter is an edge-preserving low-pass filter, well suited for depth image denoising.

4.1.2 Total Variation - L1 Filter

The TV-L1 filter is based on the observation that the noise in an image can be described by the *total variation* of a pixel’s intensity value, relative to its neighbors. Minimizing the total variation between pixels therefore reduces the noise, but may also smooth out important features of the image, such as edges and corners. In order to preserve edges, the filter output values are kept “close” to the original image pixels in I using a regularized penalty, proportional to the norm of the difference. The TV-L1 filter can then be formulated as an optimization problem:

$$\min_{u \in X} \|\nabla u\|_1 + \lambda \|u - g\|_1, \quad (4.2)$$

where u is a vector of filtered pixel values, g is a vector containing the original values from I , λ is a regularization parameter controlling how close to the observed image we wish the result to be and X is the space of attainable pixel values. Finding efficient means for solving this problem is still an active area of research, though many approaches already exist. Here we will employ a method proposed by Chambolle and Pock [79], results from which are shown in Fig. 4.2(c). The edge-preserving properties of TV-L1 filtering is owed to the L1 norm which, compared to a Gaussian filter (equivalent to a quadratic penalty) induces a higher penalty on small variations and much smaller penalties on large variations.

4.1.3 TSDF for depth image denoising

Obtaining a denoised depth image I_D from a virtual camera position c given a TSDF can be accomplished using standard ray-marching [24], as detailed in Algorithm 1 in Chapter 2. Unlike the previously discussed single frame noise filtering approaches, TSDF denoising uses all prior information and incorporates viewpoint knowledge, in order to produce a filtered image. This approach has been shown to result in reliably good estimates of depth from moving sensors[82]. We will compute two types of denoised depth images — rendered from the incrementally constructed TSDF in an online fashion (Fig. 4.2(d)), and obtained using the TSDF of the full data set retroactively (Fig. 4.2(e)) for offline applications. Both cases are relevant to several recurring tasks in robotics — ranging from object detection to place recognition.

4.2 Features on Noise-Filtered Depth

In this section we will briefly make an overview of the NARF feature detector, and several feature descriptors. We chose the following methods because of their

simplicity (NARF), reported performance (Kernel descriptors) and apparent popularity in the robotics community (FPFH), though are by no means an exhaustive list.

4.2.1 NARF feature detector

The NARF feature detector was designed for range-images. Range images are slightly different from depth images in that the value stored at each pixel location in a range image is the length of the vector to the surface, from the camera. In contrast, depth is the distance along the optical axis (by convention, simply the z-component of the surface point, in camera-centric coordinates). NARF finds regions of interest in a given range image by first locating the boundaries of objects, defined by large range differences at adjacent pixels. A score is then computed for the local variation around each pixel, compared to the variation of neighboring regions. Features are determined as the locations that are different from neighboring regions, but relatively uniform within their immediate surroundings, as this promotes repeatability in detection. Since NARF features are detected in images, it uses a concept of neighborhood defined by groupings of pixels. To avoid computations on surfaces that are far apart, the feature detection takes into account borders of objects explicitly. The image-based approach is fast to compute, since it does not require searching for neighbors in a three-dimensional region, but it makes the features sensitive to changes in silhouette and large variations in viewpoint [83].

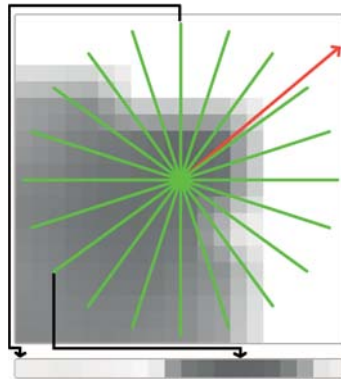


Figure 4.4: Illustration of the radial star-shaped sampling pattern used for NARF feature descriptors. The red arrow indicates the dominant direction of the patch, determined based on local curvature. The dominant direction is used to order the dimensions of the descriptor.

4.2.2 NARF feature descriptor

The NARF feature descriptor is computed at a given range image pixel by defining a patch perpendicular to the estimated surface normal. The pixel intensities within this patch are then evaluated along a pre-defined star-shaped pattern of directions radiating out from the query pixel. The pixel intensity variation along each radial direction is taken to be one dimension of the descriptor's feature descriptor. For in-plane rotation invariance, a dominant direction is also identified within the normal-aligned patch and the feature descriptor is shifted relative to this direction. An illustration of the sampling pattern used for the descriptor (source: [80]) is shown in Fig. 4.4.

4.2.3 Kernel Descriptors

Kernel descriptors define the similarity between pairs of image patches. This is done by defining relevant pixel attributes for the domain (in this case depth images). Once pixel attributes have been defined, match kernels are designed to express the similarity of two patches P and Q given these attributes, e.g,

$$K_{\text{attrib}}(P, Q) = \sum_{p \in P} \sum_{q \in Q} k_{\text{attrib}}(p, q) \quad (4.3)$$

To avoid the requirement of having pairs of patches for comparison, the problem is reformulated as matching an input patch P against a dense sampling of “would-be” candidates for Q , making it a feature descriptor, learned from data that is dependent on P only. The final step is to derive low-dimensional match kernels that approximate the original kernel functions well, by reducing the densely sampled stand-in for Q to a dictionary comprised of a finite set of basis vectors [84]. A feature descriptor is then obtained with the format:

$$F_{\text{attrib}}^e(P) = \sum_{t=1}^b \alpha_t^e \sum_{p \in P} k_{\text{attrib}}(p, u_t) \quad (4.4)$$

where u_t are basis vectors uniformly sampled over the support region of pixel attributes, b is the number of basis vectors, and $\alpha_t^e, e \in \{1 \dots E\}$ are the top E eigenvectors selected using kernel principal component analysis. In this work we evaluate three types of kernel descriptors based on different pixel attributes namely, gradient kernel descriptors which express a difference in surface gradients between patches; local binary patch (LBP) kernels, which encode and compare patterns of local depth variation; and Spin/Normal kernel descriptors which measure the difference between surface normals around a given point.

4.2.4 Fast Point Feature Histogram Descriptors

FPFH has been made popular as part of the point cloud library [60] (PCL), and is a method designed to work on *unstructured* point cloud data, with no

assumption regarding adjacency or viewpoint e.g. as implied by a depth image. An FPFH feature descriptor is computed by first performing a neighborhood search in 3D, collecting points within a region of pre-defined size. Points are then considered relative to their neighbors, along with their estimated surface normals. A set of angles are computed based on the vectors derived from the relative positions of the points and their normals. These angles are binned into a histogram, constituting what is called a Simplified Point Feature Histogram (SPFH). In a second pass, a new set of neighbors are selected for each query point and the SPFHs of these neighbors are weighted inversely to their distance to the query point. These histograms are then combined using their respective weights to form a single low-dimensional feature vector that can be used as a local descriptor of the 3D points.

4.2.5 Evaluation Methodology



Figure 4.5: The industrial robot and target environment, used as an application scenario for our evaluation.

The evaluation approach is inspired by work on visual feature detectors and descriptors in the context of object modeling [85]. We will use a two step evaluation procedure — first testing the stability of feature detectors over multiple data frames, then removing the variation caused by detector instability and evaluating the accuracy of feature descriptors under ideal detection repeatability. The results from these evaluation procedures will then be used to compare the performance of detectors and descriptors both against each other, as well as over various types of filtered depth data.

In all of the performed evaluations, we will use data-sets collected from a moving Asus Xtion Pro depth camera with a resolution of 640x480 depth pixels. The sensor was mounted on a six axis industrial manipulator, designed for offloading of shipping containers (see Fig. 4.5). During data collection, a container was filled with the sorts of goods typically found in containers —

cardboard boxes of various sizes, barrels, sacks and other miscellanea (see Fig. 4.2). The manipulator was then programmed to follow a typical unloading pattern while recording depth images, and two representative portions of the recorded image sequences were selected for use in the evaluation.

Given a sequence of depth images $z_i(M)$, we use the forward kinematics of the robot together with the tracking algorithm from Chapter 3 to obtain an accurately estimated camera trajectory by setting the forward kinematic pose as an initial guess for the pose when obtaining a new depth image, and performing the alignment. Tracking from depth images alone would have been challenging both due of the fast motion of the robot and the oftentimes uninformative content in the depth images, caused by occlusions. The forward kinematics by itself would likewise result in distortions, due to calibration errors and lack of temporal synchronization between encoder readings and depth images. The result is, however, that for every depth image $z_i(M)$ we can associate a global camera pose in the form of a homogeneous transformation matrix T_i . Since the robot's forward kinematic model provides a good initial estimate for the tracking algorithm at every frame, the trajectory estimate is virtually drift-free. Therefore, for the evaluation data-sets we can regard the estimated camera poses T_i as a ground truth input to the evaluation procedure.

4.2.6 Feature Detectors

The first type of evaluation measures the stability of detected feature locations, over a sequence of depth data $z_i(M)$. Feature detector stability is important as it ensures that feature descriptors can be repeatedly extracted at the same salient locations in different depth images. Thus, given two depth images $z_i(M)$ and $z_j(M)$ collected at two different camera poses T_i and T_j , we can evaluate the stability of a feature detector by calculating the percentage of locations that are detected in both depth frames. To calculate this ratio we need to first transform both sets of features from local pixel coordinates to their camera-centric 3D positions k_i and k_j , using the camera projection matrix. Next, we use the known camera positions to obtain the global feature point coordinates $K_i = T_i k_i$ and $K_j = T_j k_j$. Finally, we compute the ratio of features from frame i that have close neighbors in frame j over the total number of features in frame i :

$$s_{i,j}(t_k) = \frac{|\text{dist}(K_i, K_j) < t_k|}{|K_i|} \quad (4.5)$$

where $\text{dist}(K_i, K_j)$ is the Euclidean distance between each feature from K_i and it's closest neighbor in K_j and $|\cdot|$ denotes the set cardinality. This score function is parametrized on a feature association distance t_k , which signifies the maximum acceptable deviation in a feature position. In this evaluation we did not check for occluded feature locations or features that do not belong to both fields of view, and thus even a perfect detector will not achieve a score ratio of one. The

camera motion between different evaluated frames is however relatively small and occlusion effects are minimal. In addition, in this evaluation we are interested primarily in comparing feature stability over differently processed depth images and thus the feature ratio is only used as a relative and not an absolute measure. To test the sensitivity with respect to viewpoint changes, we compute the feature stability score $s_{i,j}$ over varying translation and rotation between the input depth images $z_i(M)$ and $z_j(M)$ and report stability histograms.

4.2.7 Feature Descriptors

The second type of evaluation performed in this work measures the uniqueness of different feature descriptors. In order to provide a reliable match between two feature point locations, their feature descriptor representations need to be sufficiently similar. In addition, feature descriptors extracted at different locations need to be sufficiently different, in order to provide a clear separation between matching and non-matching points. The similarity between two feature descriptors v_i and v_j is calculated using a parametric distance $\phi(v_i, v_j) = ((v_i - v_j)^T S (v_i - v_j))^{0.5}$, where S is a measure matrix that weights the importance of each dimension of the feature descriptor. Depending on the application scenario, different methods can be used to learn the matrix S and select informative dimensions. In this work we take the standard baseline approach of using an identity S matrix, resulting in a Euclidean distance measure ϕ_e .

Identifying matching feature descriptors can be achieved by simply setting a threshold on the maximum allowed Euclidean distance, but as suggested by previous works [85], this approach is suboptimal. Much better performance can be achieved by using a relative feature distance. Given two sets of feature descriptors $V_i = \{v_i^p\}_{p=1 \dots |V_i|}$ and $V_j = \{v_j^s\}_{s=1 \dots |V_j|}$, the relative distance ϕ_w between a vector v_i^p and its closest neighbor $v_j^{s1} \in V_j$ is calculated as:

$$\phi_w(v_i^p, v_j^{s1}) = \frac{\phi_e(v_i^p, v_j^{s1})}{\phi_e(v_i^p, v_j^{s2})}, \quad (4.6)$$

where v_j^{s2} is the second closest feature descriptor from V_j . The relative distance has several desirable properties — it is normalized in the range between 0 and 1, and more importantly it attains low values only if $\phi_e(v_i^p, v_j^{s1}) \ll \phi_e(v_i^p, v_j^{s2})$, i.e. only if v_i^p is much closer to its match than to any other feature descriptor in V_j .

Using the relative distance ϕ_w , we can identify the matches between two sets of feature descriptors by applying a distance threshold t_r . Before describing the details of the evaluation procedure, however, we need to extract feature descriptors from the input sequence of depth images. In a typical system, the feature points detected in the previous step will be used for determining the locations at which we extract feature descriptors. In order to obtain results independent of the quality of the feature detector, however, we need to extract

vectors at precisely the same physical locations over the depth image sequence. Therefore, we manually choose and track a set of informative feature locations and thereby decouple the feature detection and feature extraction evaluations. An example view from the manual feature definition tool is shown in Figure 4.2(f). The red crosses represent user defined feature locations, which are tracked throughout the depth image sequence and reprojected in each frame, with field of view and occlusion checks. In this manner, we also obtain reliable ground truth matching data — every manually selected feature and all feature descriptors extracted at the same physical location are globally identified.

Knowing the ground truth association between features extracted at different frames, we proceed similarly to [85]. For any two depth images $z_i(M)$ and $z_j(M)$ we calculate the numbers of correctly and wrongly matched features, depending on the feature association distance threshold t_f . By varying t_f in the range $[0, 1]$, we obtain different values for the correctly (true positive) and wrongly (false positive) associated features. In [85] the cut-off threshold is manually set to a single value for all evaluated feature extraction techniques, which can introduce some bias in the subsequent results. We use instead a standard approach for binary classifier tuning and set the threshold to a value that achieves equal precision and recall values. Essentially, at this value the number of wrongly matched features is roughly equal to the number of matches that are not reported. We calculate the percentage of correctly detected matching features, over all ground truth matches between the two frames. This value is then accumulated into two histograms, over the translation and rotation difference between the two frames. In the next section, we report our results for feature detector stability and correct match rates of different feature detectors and descriptors.

4.2.8 Results

The results for NARF feature detector stability over translation and rotation viewpoint changes are shown in Fig. 4.6. The percentage of re-observed features from different viewpoints increases by roughly 20% when the raw data is filtered using the bilateral or TV-L1 methods. The increase in performance when using the TSDF denoised images accounts for roughly another 20%, compared to the two noise filters. These results strongly suggest that the benefit of using past observations for data denoising has a significant positive impact on features stability.

A few “caveat emptor” to bear in mind that when analyzing these results, is that the camera trajectory is composed of both rotations and translations. Even though we sort the results by either rotation or translation magnitude, we do not discount the effect of the uncontrolled variable, meaning that both results remain *mixed*. Since the rotations are expressed in terms of the *camera pose* rather than e.g. pivoting at individual feature locations, the apparent motion of features will often contain large translation components, even when the camera

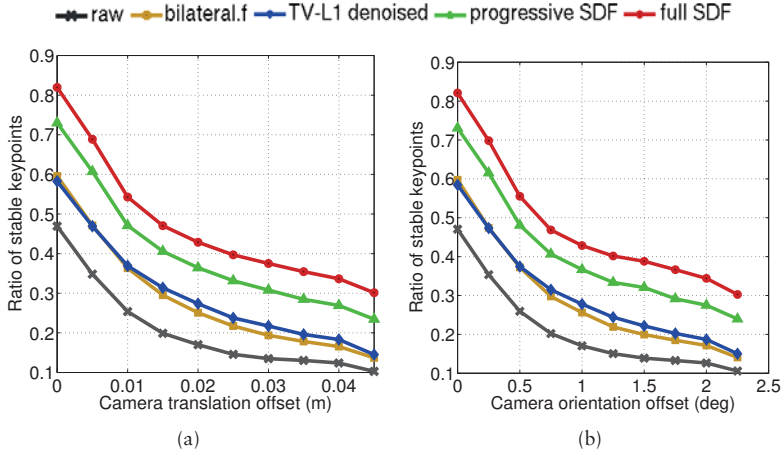


Figure 4.6: NARF feature detection, for increasing baselines in camera translation and rotation

undergoes pure rotation. The magnitude of induced translation $\|t\|$ caused by an angle change of θ in the camera orientation, for a feature point at a distance d from the sensor is equal to:

$$\|t\| = 2 \cdot d \cdot \sin(\theta/2). \quad (4.7)$$

From this equation we can infer that a change in camera angle of e.g. 3° , for a feature point at 1m distance, would result in roughly 5cm apparent translation. Since many of the objects surveyed by the robot are located approximately between 1 and 2 meters from the end-effector, this explains the observed similarity between graphs related to translations and rotations. Lastly, we should note that the graphs are a representation of an underlying histogram, for which only the center of the bins are shown in the graph. The combination of these effects explain why none of the curves have a unit score, even for apparent “zero” motion.

Results of the evaluation of the five types of feature descriptors chosen are shown ¹ in Figures (4.7—4.11). The ratio of correctly identified matches is measured as a function of both translation and orientation offset in the camera pose and shown on separate plots. The impact of reducing the noise, in terms of feature matching is vast, in some cases, such as for the FPFH descriptor leading to over four times more matches compared to the raw data, as shown in Figs. 4.8(a) 4.8(b). Across all the feature descriptors tested, the TSDF denoised depth maps show by far the largest boost in performance. It is encouraging to

¹please note the difference in scale of the ordinates

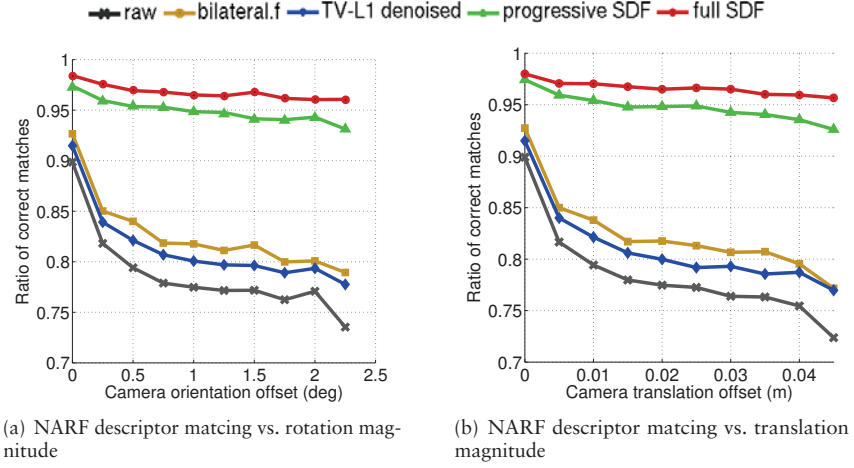


Figure 4.7: NARF descriptor matching

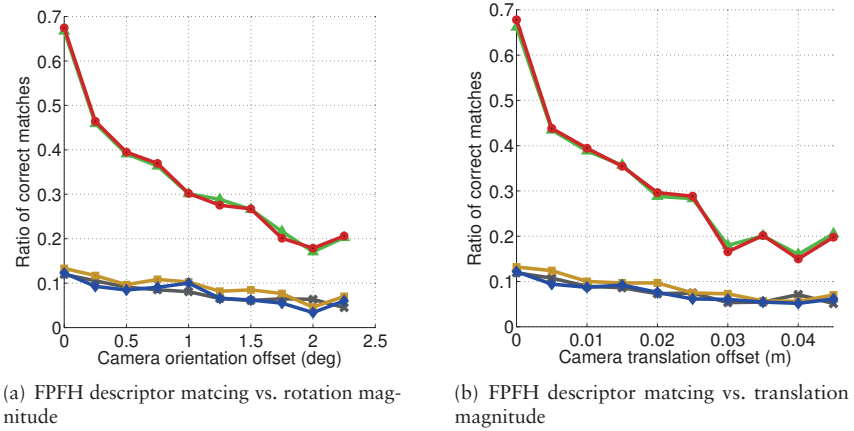


Figure 4.8: FPFH descriptor matching

see that there is little difference between the progressively denoised model, based on only past observations, and the model which incorporates the entire depth image sequence at once. Interestingly, the bilateral filter improves the matching rate on NARF feature descriptors shown in Figs. 4.7(a) and 4.7(b), but has an adverse effect on the matching of LBP kernel features, shown in Fig. 4.10(a) and 4.10(b), whereas the TV-L1 denoising yields a consistent improvement for

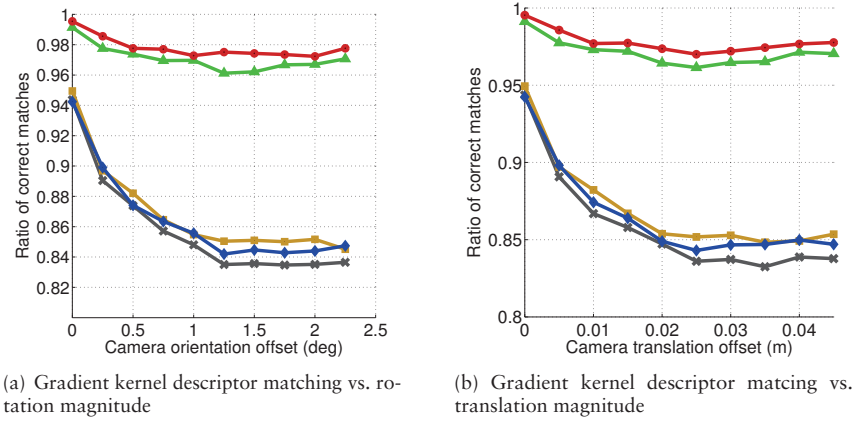


Figure 4.9: Gradient kernel descriptor matching (n.b. the ordinate axis is scaled to fit the data, and differs between the two graphs)

matching with the LBP descriptor, but performs slightly worse than the bilateral filter in other cases. When using gradient kernel features, shown in Figs. 4.9(a) and 4.9(b) neither of the single-frame denoising methods produce noticeable improvements over the raw data.

Beyond quantifying the improvement that denoising incurs in descriptor matching, we can also make a brief comparison between the different descriptors for this particular application. We note, for instance that the rate of matching for FPFH is comparatively low, possibly because it is designed to work with unstructured point cloud data rather than image patches. Also, because FPFH is a true 3D feature descriptor, it could make use of points collected from many different viewpoints, if these were consistently fused into a single frame of reference. The Gradient and LBP kernel features show the best overall performance, on both raw and denoised data. The NARF feature descriptors, simple as they may be to compute, show a remarkably good performance. The spin kernel descriptors appears to be the most sensitive to viewpoint variation, though it has among the highest matching rates for short baselines, as shown in Figs. 4.11(a) and 4.11(b).

4.2.9 Discussion

The evaluation presented above clearly indicates the benefit of computing local shape descriptors on more consistent and less noisy depth images. It would be interesting to compare the approach to other noise filtering techniques that use model-based priors, such as e.g. planes [86], wavelets [87], curvelets [88]

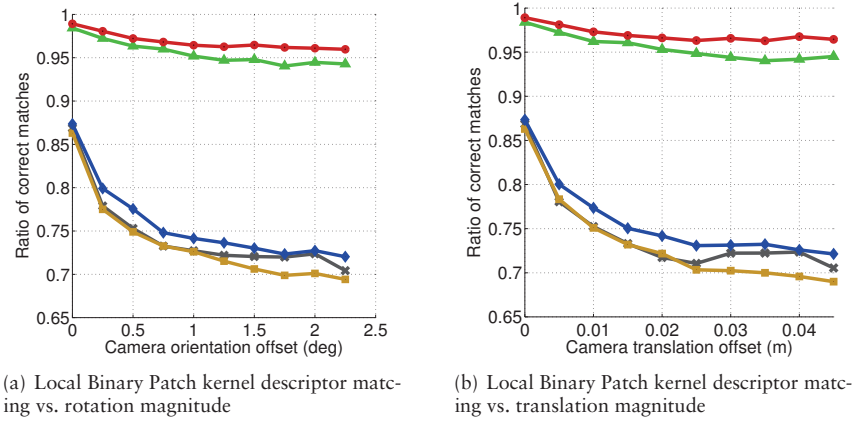


Figure 4.10: Local Binary Patch kernel descriptor matching

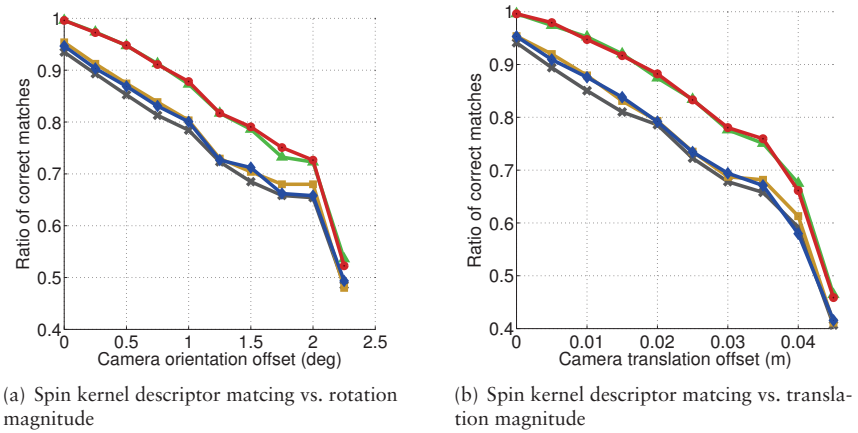


Figure 4.11: Spin kernel descriptor matching

or any of the many other edge-preserving noise removal methods currently available [89]. We have seen that shape-based feature detectors lead to better repeatability when computed on denoised depth images. For all the edge-preserving noise removal methods we studied, we were able to observe that features were being detected at more consistent locations, in spite of viewpoint transformations. The gap between single-frame noise-removal and pose-aware TSDF denoising in feature detection performance is quite large, but in any case,



Figure 4.12: a 3D shape, optimized to have different silhouettes when viewed from orthogonal directions. Discontinuities in the shape are therefore completely different depending on viewpoint and thus likely to result in disparate descriptors, even in depth-based imaging.

some form of edge-aware denoising should be considered as a step prior to feature detection. We also saw that the descriptors in some cases benefit from single-frame denoising but were *always* more distinctly matched when using the TSDF for filtering.

If only a single view-dependent snapshot of the geometry is to be used for object detection and recognition, it makes sense to strive for the best possible version of that snapshot, for the task at hand. However, there are inherent limits on the distinctiveness of shapes given their projections. As illustrated well by the Shadow Art [90] algorithm, geometries such as the one depicted in Fig. 4.12² are entirely possible and the descriptor statistics for such shapes, seen from different views may well be multi-modal. Side-stepping the issues related to describing 3D geometry from their projected depth images is the topic of the next section, where we study feature detection in volumetric representations directly.

4.3 3D Feature Detection

An issue that remains with feature detectors and descriptors operating in depth images is that they are bound by the limits of observability. By this we mean that any projective camera can only see, at most, half of an object (exactly half, in the case of an orthographic camera³). This is illustrated in Fig. 4.13, showing two different objects that appear identical to the sensor due to self-occlusions. In the image, some pairs of rays emitted from the sensor have been highlighted in blue. Although the angle between each of the rays forming a pair is constant, the length of surface between the points where the ray intersects the object is variable (shown in red). Since the rays also diverge as one moves further away from the sensor, the points of intersection between rays and surfaces also spread further apart, even for surfaces without curvature. The variable sampling frequency caused by projective cameras and occlusions are two motivating factors for avoiding a projective space, if possible. In this section we shall therefore look at some feature detectors, defined in a metric 3D space, where neighboring

²source: https://graphics.stanford.edu/~nilyoy/research/shadowArt/shadowArt_sigA_09.html

³e.g. using a telecentric lens or a flatbed scanner

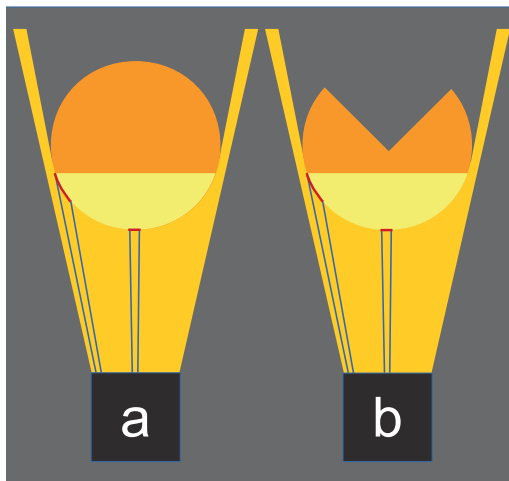


Figure 4.13: Illustration of the limits of observability with a projective camera; the shape appears identical to cameras *a* and *b*

samples in the data are also neighbors in the spatial sense. In the following sections, we will introduce three different feature detectors and the theory behind how their response functions are defined. In Section 4.3.1 we describe the volumetric application of the Harris corner detector, which is traditionally computed in the gradient space of two-dimensional intensity images. Because the Harris feature is gradient-based, we dedicate some attention in Section 4.3.2 to discuss the rationale behind different choices of gradient estimators and explain their derivation. Taking an alternative route to feature detection in TSDFs, in Section 4.3.3 we review the concepts of integral invariant features whose defining characteristics are that they require no gradient estimation and have been proposed as a fast feature detector for signed distance fields. We follow this up with an evaluation of detector stability and a discussion of the implications of the experimental results.

4.3.1 Harris Corners

Most feature detection methods apply a response function over the entire image domain and retain the locations for which the function both exceeds a threshold and is also locally maximal. One such response function is based on the sum of squared differences (SSD) [91] between pixels within a region around a candidate location. It is designed to give a high score to points where the

image derivative is not small in any given direction, which is indicative of high curvature. Harris [92] approximates the Hessian of the SSD as

$$H_2 = \frac{1}{|w|} \sum_w \begin{bmatrix} \nabla I_x^2 & \nabla I_x \nabla I_y \\ \nabla I_x \nabla I_y & \nabla I_y^2 \end{bmatrix} \quad (4.8)$$

where w represents a square (or round, if desired) window around the candidate point, $|w|$ is the number of elements in w , for normalization and ∇I_x , ∇I_y are the estimated gradients of the image in the horizontal and vertical directions, respectively. The extension to 3D is straightforward.

$$H_3 = \frac{1}{|w|} \sum_w \begin{bmatrix} \nabla I_x^2 & \nabla I_x \nabla I_y & \nabla I_x \nabla I_z \\ \nabla I_x \nabla I_y & \nabla I_y^2 & \nabla I_y \nabla I_z \\ \nabla I_x \nabla I_z & \nabla I_y \nabla I_z & \nabla I_z^2 \end{bmatrix} \quad (4.9)$$

The response R for a given pixel is a measure of both the scaling and evenness of the distribution of gradients around that pixel. This is captured by the determinant and trace of H

$$R = \det(H_2) - k \operatorname{Tr}(H_2)^2 \quad (4.10)$$

and equivalently, for a voxel,

$$R = \det(H_3) - k \operatorname{Tr}(H_3)^3 \quad (4.11)$$

with k being an empirical constant for which a typical value (in the volumetric case) is 0.001 [93]. The above formulation is algebraically equivalent to the following, using eigenvalues.

$$R = \prod_{i=1}^{\dim} (\lambda_i) - k \left[\sum_{i=1}^{\dim} (\lambda_i) \right]^{\dim} \quad (4.12)$$

Since the detector is based on a computation carried out over a neighborhood, many adjacent points will have similarly high response functions. To enforce the property of being *locally maximal* i.e. avoiding *clusters* of detected features, non-maxima suppression is used. Non-maxima suppression entails iterating over all candidate features within a local neighborhood, keeping only the single highest-scoring.

4.3.2 Derivatives

A point to be made against response functions based on gradients is that gradients are susceptible to noise and that this in turn reduces the stability of the resulting detectors. Image derivatives may be obtained by a simple central differencing scheme but are often calculated by convolution with a filter kernel that

represents the weighted average of several central difference computations. By including pixel samples from a neighborhood around the point of interest, some robustness to noise is obtained at the cost of performing more computation and reducing the locality of the estimate. The same applies to voxels. A common choice of filtering kernel in 2D is the 3x3 Sobel-Feldman operator [94] which can be interpreted as the application of a low pass filter (an integer approximation to the Gaussian kernel) and differentiation. See Eq. (4.13) for the example of the derivative filter in the horizontal direction where $*$ denotes a 2D convolution or equivalently, (4.14) using ordinary matrix multiplication.

$$\text{SoFe}_h \in \mathbb{R}^{3 \times 3} = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad (4.13)$$

$$\text{SoFe}_h \in \mathbb{R}^{3 \times 3} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad (4.14)$$

By convolving the differencing and blurring operators with themselves, i.e.,

$$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \end{bmatrix} \quad (4.15)$$

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} \quad (4.16)$$

one obtains filter coefficients that can be combined, in the same manner as in Eq. (4.14) to produce a 5×5 derivative kernel. Generating the volumetric filter kernels from the 1D coefficient vectors is analogous to the 2D case and is detailed in Algorithm 2. The variables g , b , direction , n are column vectors containing the derivative and blurring filter coefficients, derivative direction and kernel size (3 or 5), respectively. The derivative and filter coefficients appear in Table 4.1, for reference.

Since the Sobel-Feldman operator is an approximation to the derivative of the Gaussian probability density function, the latter is worth some consideration as an option, too. For our analysis, we compute the analytic derivatives of the Gaussian and directly form the 3D filter kernels [95] of size $3 \times 3 \times 3$ and $5 \times 5 \times 5$ from Gaussian with variances of $\sigma_3 = 0.95$ and $\sigma_5 = 1.25$, respectively.

An additional option well worth considering is the Scharr operator [96]. This is a derivative kernel arrived at by numerical optimization seeking to maximize rotation invariance by minimizing the squared angular error in the Fourier domain. As with the derivative-of-Gaussian it too has non-integer filter coefficients.

Algorithm 2 Computing the volumetric filter kernels from their 1-D coefficient vectors

Require: $b, g, \text{direction}, n$

```

1: Allocate  $n \times n \times n$  elements for  $K$ 
2: switch (direction)
3:   case "x":
4:      $S \leftarrow bg^T$ 
5:     for all  $z$  in 1 to  $n$  do
6:        $K_z \leftarrow s_z f^T$  {  $K_z$ , is the  $z$ -th slice of  $K$  and  $s_z$  is the  $z$ -th column of  $S$  }
7:   case "y":
8:      $S \leftarrow gb^T$ 
9:     for all  $z$  in 1 to  $n$  do
10:       $K_z \leftarrow s_z b^T$ 
11:  case "z":
12:     $S \leftarrow bb^T$ 
13:    for all  $z$  in 1 to  $n$  do
14:       $K_z \leftarrow s_z g^T$ 
15: end switch
16: Return  $K$ 

```

name	coefficients
SoFe ₃ derivative	$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}^T 1/2$
...filter	$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T 1/4$
SoFe ₅ derivative	$\begin{bmatrix} 1 & 2 & 0 & -2 & -1 \end{bmatrix}^T 1/6$
...filter	$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}^T /16$
Scharr ₃ derivative	$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}^T 1/2$
filter	$\begin{bmatrix} 46.84 & 162.32 & 46.84 \end{bmatrix}^T 1/256$
Scharr ₅ derivative	$\begin{bmatrix} 21.38 & 85.24 & 0 & -85.24 & -21.38 \end{bmatrix}^T 1/256$
...filter	$\begin{bmatrix} 5.96 & 61.81 & 120.46 & 61.81 & 5.96 \end{bmatrix}^T 1/256$

Table 4.1: Filter coefficients used to derive 3D derivative kernels

4.3.3 Integral Invariant Features

Integral invariants were first introduced by Manay et al. [97] and are local shape descriptors defined as integrals over a rotationally symmetric neighborhood. They propose a *local area invariant* and *distance invariant* that both result

in efficient and noise-robust estimates for *mean curvature* of a shape in 2D. Pottmann et al. [98] presented integral invariants defined on three-dimensional signed distance fields and proposed an extension of the local area invariant to the volumetric case. Here we further extend the study of *signed distance invariants* and *volume invariants* to their application on truncated signed distance fields. Both of these features have their domains defined as the volume bounded by a sphere, centered at a *surface* point p . The assumption that computation is carried out centered on surface points implies that voxel-based methods are a poor fit, since the probability of a voxel being exactly on the surface i.e. the zero-level of the TSDF, is very small. However, Pottmann et al. [98] mathematically show that these features are robust to perturbations of the query point location, if the integration radius is sufficiently large. This reported stability encourages our attempt to apply integral invariants even in the discrete case.

The volume invariant $V_r(p)$ is the integral of the indicator function $1_D(x)$ which returns 1 if x is in occupied space, and 0 otherwise. This information can be obtained from the TSDF by testing the sign of the field at any given point (negative if occupied, positive otherwise). The signed distance invariant, $D_r(p)$, is simply the integral of the signed distance field. Both of these feature detectors are defined within the bounding sphere of radius r . Formally,

$$V_r(p) = \int_{p+rB} 1_D(x) dx, \quad (4.17)$$

$$D_r(p) = \int_{p+rB} \text{dist}(x, \Phi) dx \quad (4.18)$$

The features are illustrated in Fig. 4.14 and Fig. 4.15, respectively. The mean curvature of the surface is estimated by computing the difference between the result of the integration (or summation, in the discrete case) and the result which would have been produced if the computation had been carried out on a perfectly planar surface. The following expressions approximately relate the mean curvature estimates $\tilde{H}_v(p)$ and $\tilde{H}_d(p)$ of the surface to the respective descriptor values.

$$\tilde{H}_v(p) = \frac{8}{3r} - \frac{4V_r}{\pi r^4} \quad (4.19)$$

$$\tilde{H}_d(p) = \frac{15D_r}{4\pi r^5} \quad (4.20)$$

From the above equations we note that the estimated mean curvature for volume integrals is zero if (and only if) the amount of occupied space is equal to half of the sphere, i.e. the curvature estimate is an affine function with a specific reference point. The equation based on the signed distance integral is simply linear. As such, the signed distance integral relates mean surface curvature to the amount of imbalance in the total positive and negative fields on either side. A downside of not using first-order (gradient) information about the field becomes

apparent here, as there is no way to distinguish saddle points from flat surfaces, since the mean curvature is zero in both cases.

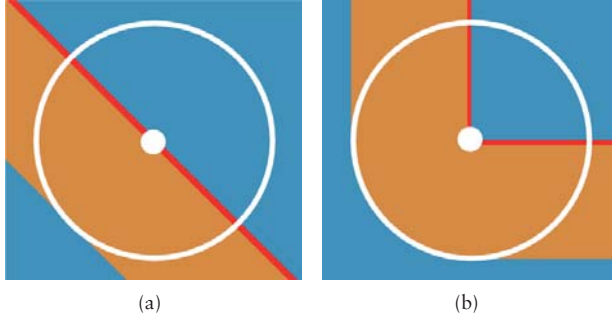


Figure 4.14: Illustration showing the regions in which the indicator function $1_D(x)$ would return 1 (orange) and 0 (blue).

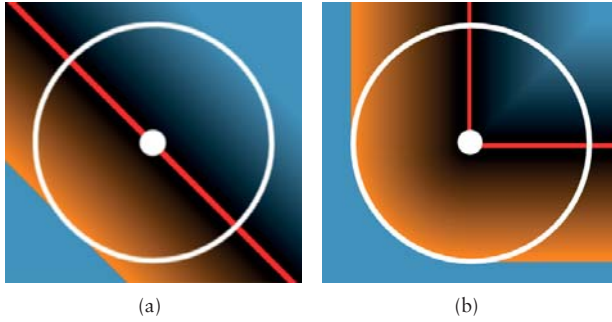


Figure 4.15: Illustration of the signed distance invariant at a flat surface and a corner. The surface interface (i.e. the zero-distance iso-level) is represented by a red line. Orange indicates a negative value. The sum of all the values enclosed by the sphere results in a negative value for the corner and in zero for the plane

4.3.4 Evaluation Methodology

We are interested in evaluating how repeatable the feature detectors are. This interest is motivated by the cases where a robot returns to previously visited locations (or observes known objects). In such scenarios, it would be ideal to extract geometric descriptors at the exact same places as before, since this would

yield the highest amount of matching local descriptors and allow the robot to be more certain of where it was and what objects were present in the environment.

In general, re-mapping the same environment or objects will produce slightly different results each time. Among the factors that prevent the acquisition of identical maps are differences in measurements from the sensor, variations in pose estimation when integrating the data, and changes in the alignment of the voxel grid. To simplify our analysis, we will only consider the robustness of the feature detectors with respect to changes in the alignment between the initial pose of the voxel grid (modulo the voxel size) relative to the sensor. We shall see that this alone has a substantial impact on repeatability, as it accounts for both sample aliasing in the grid and anisotropy of the feature detectors.

To ensure that the sensor data and estimated trajectory are not a source of variation, we use a pre-recorded data sequence with a globally optimized trajectory [99] and reconstruct the environment using the volumetric integration strategy laid out in Chapter 2. At the start of each reconstruction, we transform the initial pose of the camera relative to the voxel volume by increasing amounts of translation and rotation. At the end of each session, the different types of features are extracted and we count the number of features that remained stable in proportion to the total amount. Defining Q_s to be the set of features locations in the unmodified or source configuration and Q_t to be the set of features locations extracted from the target volume, for which the camera pose was initialized with a transformation T_0 . Let $\dot{q}_s \in Q_s$ and $\dot{q}_t \in Q_t$ denote homogeneous vectors in \mathbb{R}^4 , e.g. $[x, y, z, 1]^T$, and $T_0 \in \mathbb{R}^{4 \times 4}$ a transformation matrix including rotation and translation and $|\cdot|$, the cardinality operator. We then define stability as the average between source to target and target to source matches, where a match is determined to have occurred if two features are within $\tau_{\text{match}} = 2$ voxels of each-other.

$$\text{score} = \frac{1}{2(|Q_s| + |Q_t|)} (|\{\forall q_s, \min_{q_t} \|\dot{q}_s - T_0^{-1} \dot{q}_t\|_2 < \tau_{\text{match}}\}| + |\{\forall q_t, \min_{q_s} \|T_0 \dot{q}_s - \dot{q}_t\|_2 < \tau_{\text{match}}\}|) \quad (4.21)$$

Our definition of the matching score avoids being overly generous or strict in case the amount of features differ between the two sets by checking for corresponding features in both directions. We compute the matching scores for varying baselines in translation and rotation:

- translations offsets of $\frac{1}{8}, \frac{2}{8}, \dots, 1$ voxels are applied combinatorially along all dimensions. The sub-voxel shifts are justified by the fact that translating the volume by whole voxel increments does not alter the aliasing and sampling issues that we wish to investigate.
- rotational offsets of $\frac{1}{8} \cdot \frac{\pi}{4}, \frac{2}{8} \cdot \frac{\pi}{4}, \dots, \frac{\pi}{4}$ degrees are also applied combinatorially, around each principal axis. The reason for the chosen interval is that all the algorithms involved are symmetric along the principal axes. Any larger rotations than $\frac{\pi}{4}$ could therefore be achieved by a smaller one and a

transposition of the appropriate dimensions (which would not affect the results).

The repeatability score is computed for each reconstruction and detector and binned together by the offset relative to the default pose. Because the translations were performed in each of the dimensions separately, i.e., in a *manhattan-like* way, plotting the scores against the L_1 norm of the offsets yields a better-looking graph. For rotations, we compute the equivalent angle-axis parametrization and bin the results by the magnitude of the angle.

4.3.5 Experimental Results

The following analysis is based on a volumetric integration of the *copyroom* data sequence⁴ with voxel size $v_{\text{size}} = 0.015\text{m}$, and truncation at $\pm v_{\text{size}} \times 4$. We use a sufficient number of voxels in each dimension to allow the entire reconstructed surface to fit in the volume in spite of large rotations, and thus avoids problems with bookkeeping of out-of bounds features. The relatively large truncation distance is chosen to give the integral invariant features a better chance at producing stable features. For each reconstruction session we use the first 2500 frames, as this captures representative parts of the scene with diverse characteristics. For all experiments, we set the non-maxima suppression window to be of size $7 \times 7 \times 7$ and set thresholds for culling features for which the response function is low.

Gradient-based methods

We show the performance of Harris features, when computed using different gradient estimation methods in Fig. 4.16. In all cases the window size w over which the summation of the gradients to form H (see Eq. (4.9)) is carried out was set to $5 \times 5 \times 5$. The smaller $3 \times 3 \times 3$ kernels all produced slightly worse results than their larger counterparts though with similar trends, we omit them for clarity of presentation. We find that the derivative of Gaussian outperforms Sobel-Feldman which in turn outperforms Scharr kernels. For all three, increasing the rejection threshold results in a larger proportion of stable features.

Central differences are cheaper to compute but offer poor repeatability, and when the feature rejection threshold is increased a larger proportion of high quality features are culled, noted by the drop in repeatability.

The sensitivity with respect to rotation is shown in Fig. 4.18. Note that since the volume is not pivoted around the feature locations but around the camera origin, some translation is induced as well. Although the Scharr kernel produces the least amount of variation with respect to rotation, the repeatability of Harris features is higher when gradients are computed based on both Sobel-Feldman

⁴available from <http://qianyi.info/scenedata.html>, and given a proper introduction in Chapter 6

and derivative of Gaussian kernels. Central differences provide the least robust gradient estimate, under rotation, as expected.

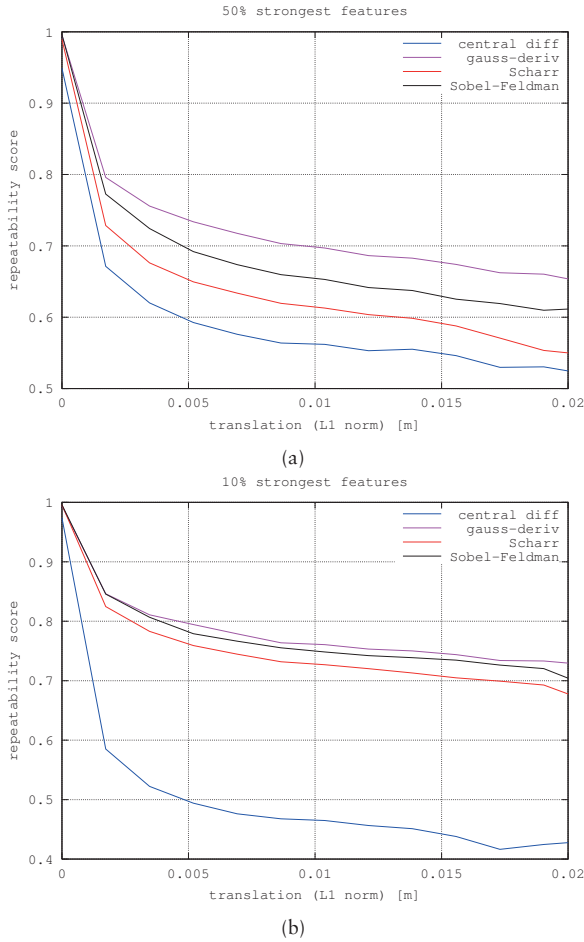


Figure 4.16: Repeatability of Harris features computed with the derivative of Gaussian kernel (size $5 \times 5 \times 5$) when threshold is set such that only the top (a) 50% and (b) 10% features are maintained

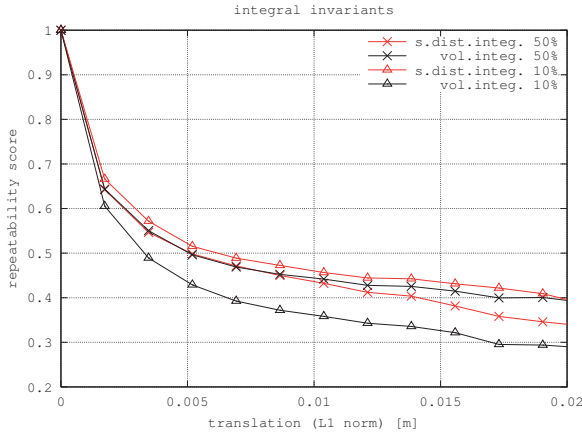


Figure 4.17: Repeatability of integral invariant at different threshold levels, with respect to translation

Integral-based methods

The integral based methods, namely the volume integral and signed distance integral features, do not perform as well as the gradient-based method. We see in Fig. 4.17 that their repeatability is below that of all variants of Harris features. Increasing the rejection threshold does not produce much improvement for the signed distance integral, and causes a slight deterioration in the case of volume integrals. The rotation invariance, shown in Fig. 4.18 is good, in spite of the spherical integration region being a discrete approximation with a radius of 3.5 voxels. The diameter of the integration region is made to match the truncation distance of ± 4 voxels. However, the actual width of the non-truncated region of the TSDF may both be larger, depending on sensor noise, and smaller, due to surfaces being oriented (as defined by their normal direction) at near-perpendicular angles relative to the line of sight to the sensor. The dependency between integration radius and truncation distance is more critical for volume invariants, where the signed distance field should ideally not be truncated within the radius of the integration region. This is because the volume integral is compared to a specific reference value for curvature estimation, and this reference would need to be readjusted in case half of the volume being occupied, as seen in Fig 4.14(a) no longer corresponds to a planar surface.

Additionally, for signed distance invariants the thickness of the thinnest object should be at least twice as large as the radius of the integration region. This is due to the balance needed between positive and negative fields to indicate a mean curvature of zero. In the case of thin objects, the negative side of the

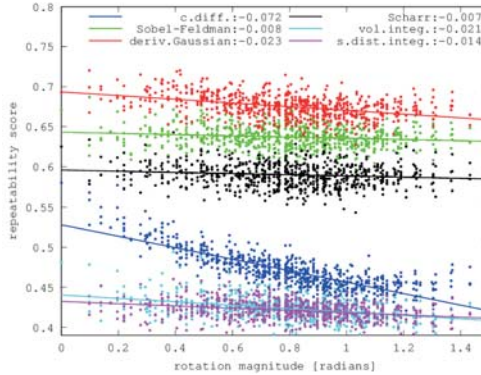


Figure 4.18: Repeatability of Harris using different derivative kernels, in addition to integral invariant features. The factors in the legend indicate the slope of the linear approximation to the data.

distance field will only decrease until the center of an object, leading to skewed estimates.

To highlight another problem of integral invariants applied to truncated distance fields, consider Fig. 4.19. Indicated in green is the amount of additional occupancy caused by a sharp concave bending of the surface. Here we see that increasing the radius of the integration region beyond the truncation distance adds no useful information to the curvature estimation. In fact, it only serves to reduce the relative difference between curved and planar surfaces, compare the case of Fig. 4.19 with those of Fig. 4.14 for example. When the distance field extends at least as far as the circle radius, a 90° bend results in a whole quarter of the circle area being added to the feature response. However, when the TSDF is represented by a thin band around the surface, the added area that arises when the surface is bent, is relatively small compared to when the surface is flat. The results show that the signed distance integrals fare slightly better than the volume integrals which is expected since the former use both the positive and negative regions of the field and therefore have a slightly larger sample base. However, neither is large enough to robustly filter out noise at the tested radius. Since merely extending the radius is fruitless, we are left with the option of extending the TSDF truncation distance to accommodate a larger integration region. However, this leads to a decrease in the quality of scene reconstruction and makes the reconstructed geometry more dependent on the sensor's viewpoint locations.

The main advantage of integral invariant features is that they are relatively cheap to compute, requiring only a sum of the distance field within a bounding region. To make the computational cost more tangible, a naive GPU implemen-

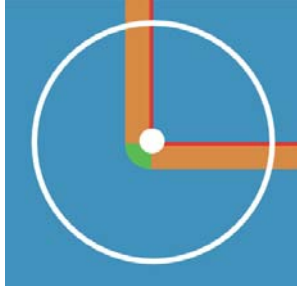


Figure 4.19: 2D analogy of volume integral with occupancy estimated on a narrowly truncated signed distance field. The green region shows the total additional 'volume' that appears due to a sharp right-angled bend

tation⁵ of derivatives by convolution with a filter kernel takes between 680ms and 150ms for kernels of $5 \times 5 \times 5$ and $3 \times 3 \times 3$ respectively, for every element within a TSDF volume of 512^3 voxels. Computing the Harris features adds another 100ms, with approximately 8ms more for the non-maxima suppression. In contrast, the integral invariants take around 20ms to compute, followed by the non-maxima suppression adding to a total of 28ms.

4.4 Discussion

Feature detection is a powerful tool that can allow for great savings in computation, by allowing other algorithms further down the pipeline to focus on a smaller subset of data for processing. To be useful, the feature detector should be repeatable, ideally flagging the same data as features every time they are observed. However, in our experiments we have seen that features detected on depth images are quite sensitive to viewpoint variations.

A possible explanation for this sensitivity is that although the scale of the environment is not ambiguous due to the metric nature of range-sensing, depth-cameras are still subject to the peculiarities of projective geometry. This means that any fixed-sized support region used to compute features will correspond to different actual surface areas in the scene, since measurement rays diverge as they leave the sensor. Since geometric features also tend to be located on regions of extreme curvature, changes in viewpoint are most likely to produce self-occlusions or non-uniform variations in measurement quality there. A further impediment to reliable feature detection is simply the presence of measurement noise, for which denoising through a TSDF provides an effective countermeasure. Avoiding the scale issues is possible by performing the feature detection, too, in the volumetric TSDF space. However, this has its drawbacks when computing features throughout the entire volume since the number of voxels present in a

⁵i.e. not exploiting separability into a series of 1-dimensional convolutions nor exploiting sparsity

TSDF typically exceeds the pixel-count in a single depth-image, which results in substantially higher computational costs.

Chapter 5

Compression

In Chapter 2 we mentioned the memory requirements of TSDFs as one of the drawbacks for their use in robotics. The scalability of TSDFs in terms of memory is an issue that affects all voxel-based representations. For TSDFs in particular, the common method used to incrementally update the field requires storing a weight as well. An important question is therefore how the memory requirements for TSDFs can be reduced. The most straightforward approach is to simply reduce the number of bits per voxel, but this leads to quantization effects that become visible in terms of surface smoothness. The absolute limit is reached when the signed distance is represented with a single bit. At the limit, a zero or one simply indicates whether the voxel is behind or in front of the surface. Simply reducing the number of bits is an approach with limited usefulness however, since the effective voxels, i.e., voxels that can contain distinct numerical values, also become fewer as the number of bits is reduced. For a given number of bits, and depending on the combination of truncation limits and grid resolution, the steps between representable distances may be in excess of the length of an individual voxel. For applications where larger voxels are acceptable, but where the variance of the range-sensing equipment is not proportionally larger (meaning that the truncation limits need not be increased) it makes sense to attempt a lower-bit representation as a first step.

In practice, one may use a *bit field* to store the distance and weight (and perhaps even an indexed color value!) in a given number of bits or manually pack data into a byte (or bytes), as this is generally the smallest addressable unit on modern computer architectures. In the C programming language, a bit field can be defined as e.g:

```
struct voxel{
    unsigned D : 4;
    unsigned W : 4;
};
```

This declares a structure called “voxel,” containing two unsigned integers, represented with 4 bits each. One may additionally force the compiler to ensure that this structure is aligned to one byte by issuing directives such as `#pragma pack(1)` thus taking up no more memory than necessary. However, explicitly packing data into memory may cause slower memory access, since load instructions that read variables stored across the alignment boundaries require more operations. The unsigned integer stored in `D` can then be used to represent values in the range between D_{\min} and D_{\max} . For an implementation example, see code listing A.1 in the Appendix.

When deciding how to allocate the bits to represent either the weight or the signed distance one should be mindful of the fact that a single weighted update cannot effect a change in the stored value, unless the absolute value of the update exceeds the gap between two representable digits. Having more bits used to represent the distance produces better predictions of the surface interface, but with fewer bits representing weight the result tends to be more dependent on the last few samples. Low weights can make the map susceptible to corruption by spurious outliers, too. A priori, splitting the number of bits evenly between distance and weight, e.g. assigning one word/byte/nibble to each is recommended, and then measure the performance obtained when changing the balance between them.

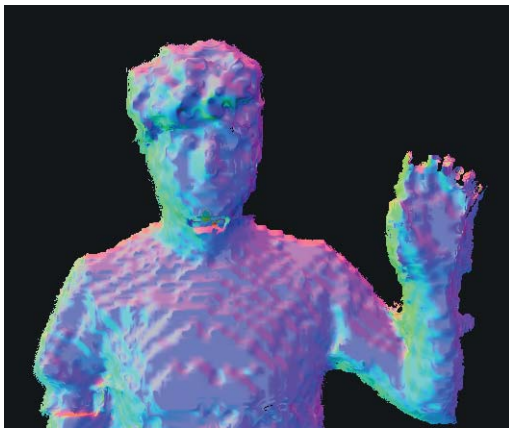


Figure 5.1: Reducing the number of bits in the distance field causes visible quantization artifacts, shown here is a representation using one byte per voxel.

Reducing the number of bits in the representation is sometimes not a viable option. Shown in Fig. 5.1 is the ray-marched TSDF using a single byte per voxel, with four bits for weight and four bits for distance. It is quite evident by the flat and blocky appearance that the number of bits used to encode the distance field is insufficient to place the zero crossing at an exact location within a voxel, even

with interpolation. The quantization effect seen here illustrates that the width of the non-truncated region of the distance field should be compatible with the number of possible values that can be represented. Ensuring that the number of representable values is in the same order of greatness as the number of voxels to store them merely implies that adjacent voxels in the varying direction of the distance field do not contain identical values, but it yields no accurate sub-voxel positioning of the surface interface when interpolating.

As awful as the surface rendering may look, we find that we are still able to track the camera with some success as far as down to a 3-bit representation, as seen in Figs. 5.2, 5.3, 5.4. At that level of quantization, the number of distinct values used to represent the distance field from D_{\min} to D_{\max} are only 7. Given that the distance field was truncated at the fixed values of $\pm 30\text{cm}$, and the voxel size was of 2cm , this results in a piecewise constant representation of the distance, since several adjacent voxels take on the same value. We find that the tracking fails more often with fewer bits, leading to significantly lower performance in trajectory estimation. However, we find that the relative error between consecutive frames does not increase as much. This indicates that reducing the number of bits leads to an increased brittleness, causing few but devastating failures. There is little difference in performance, from a reliability standpoint, when using more than 4 bits for representing the distance, and tracking will succeed and fail to the same extent. However, we can see that the average relative errors and variance of the error increase slightly each time one reduces the number of bits per voxel.

Reducing the number of bits, allows a larger number of voxels to fit into the same amount of memory but as a result, we see that the surface quality deteriorates, as does the reliability of pose estimation relative to the map. If we are interested in using the map for object detection, especially with gradient-based feature detectors and descriptors, we ought to be weary of any additional sources of high-frequency noise e.g. quantization errors. But what are the alternatives for large scale mapping using TSDFs?

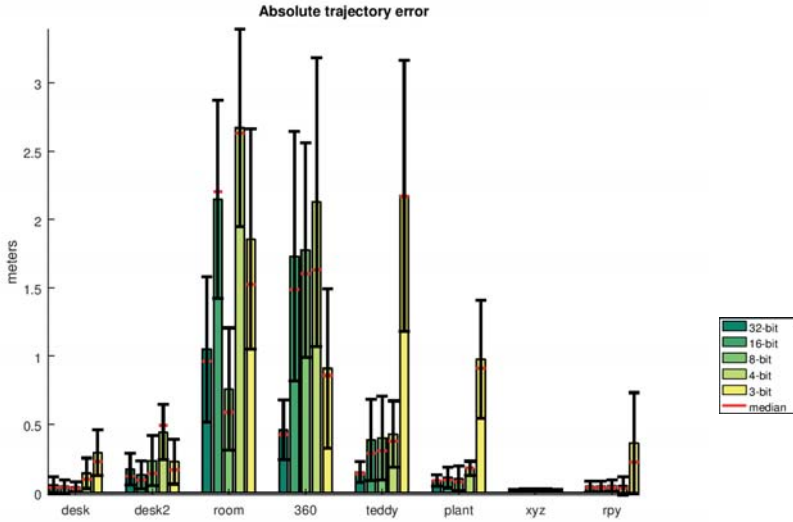


Figure 5.2: Absolute Trajectory Error for tracking and mapping, using different numbers of bits per voxel to represent the TSDF. The bars are grouped by data sequence, and their heights show the mean error relative to the ground-truth trajectory. The median pose error is represented by a red line and the variance is given by the whiskers. We see a general trend towards greater errors and variances as the number of bits is reduced, however this progression is not always monotonic for all data-sets. Failures to track the camera may lead to gross misalignments that cause all future poses to be erroneous. The large errors are generally indicative of the severity or frequency of such failures.

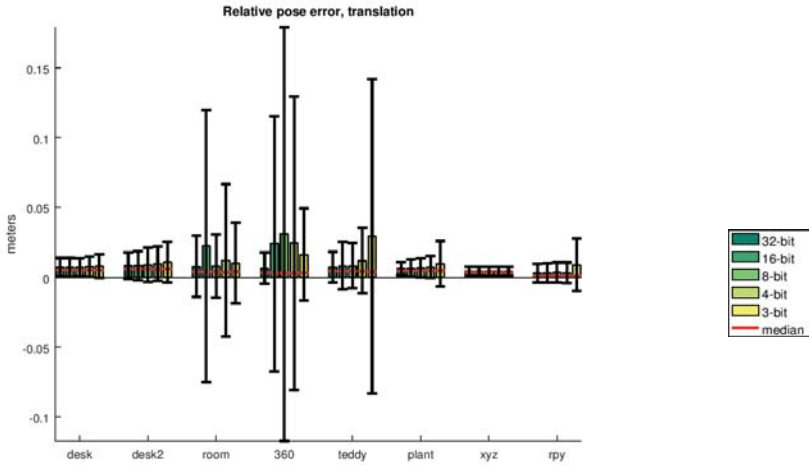


Figure 5.3: Relative Pose Error in translation, measured between consecutive frames when using different amounts of bits per voxel to represent the TSDF. The bars are grouped by data sequence, and their heights show the mean frame-to-frame error in translation compared to that of the ground-truth trajectory. The median error is represented by a red line and the variance is shown by the whiskers. The progression seen here is gradual, with errors becoming, on average, slightly larger as the number of bits is reduced. The median error is close to the mean, except for the data-sets where large tracking failures likely occur: room, 360 and teddy.

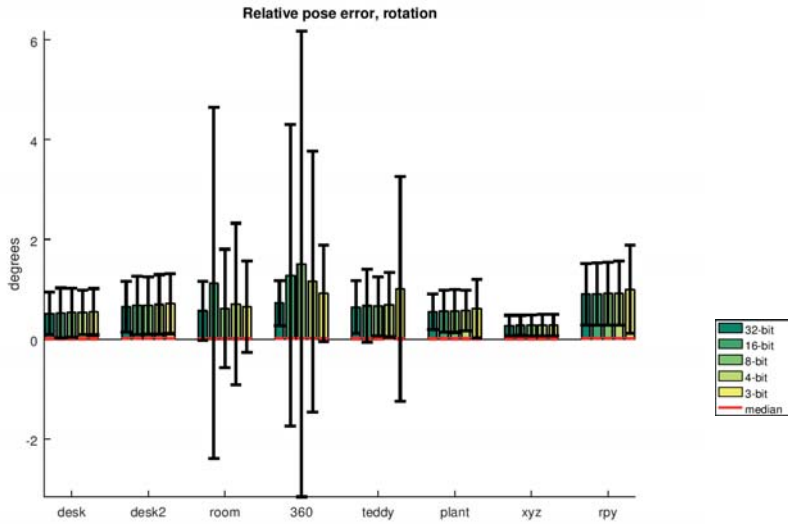


Figure 5.4: Relative Pose Error in rotation, measured between consecutive frames when using different amounts of bits per voxel to represent the TSDF. The bars are grouped by data sequence, and their heights show the mean frame-to-frame error in rotation compared to that of the ground-truth trajectory. The median error is represented by a red line and the variance is shown by the whiskers. The progression seen here is gradual with errors becoming, on average, slightly larger as the number of bits is reduced. The median error is consistently close to zero indicating that the distribution of errors is skewed by a low number of very large errors

5.1 Managing memory complexity - Related Work

5.1.1 General Purpose Compression

If throwing away bits is not feasible, the next obvious general-purpose approach that can be applied is compression. There are a number of data-compression algorithms designed for directly compressing volumetric data. Among these we find video and volumetric image compression [100, 101], including work dealing with distance fields specifically [102]. Although these methods produce high-quality compression results, they are inherently sequential in their design, and thus hard to speed up for real-time use. This is due to the complex interpolation and extrapolation methods used to reduce the amount of data that needs to be explicitly represented. One of the simplest, yet very efficient forms of TSDF compression is a straightforward run-length encoding scheme [20] with reported lossless compression ratios of 10:1 to 20:1. The common characteristic shared by these methods is that compression and decompression become inherently non-local operations.

5.1.2 Space Partitioning

Most solutions to managing the large memory footprint of TSDFs exploit the sparsity i.e., a predominance of empty regions in the environment. A common solution is found by hierarchically partitioning the volume in such a way that homogeneous regions are not represented in a redundant manner. An octree [103] is one encoding that recursively subdivides a volume into 8 axis-aligned equal-sized partitions until the resulting sub-volume is homogeneous (within tolerance limits) or until a maximum depth is reached. Since the distance field value is typically trilinearly interpolated, the optimal measure for homogeneity is not necessarily when the contents of the sub-volume is constant, but sufficiently well approximated by a linear function [104]. It has been shown that octrees can be efficiently implemented on GPU architectures [105] and applications of octrees to systems using TSDFs have claimed performance boosts in TSDF update speeds, ray-marching, and storage efficiency [106]. The use of octrees has made the maintenance of larger TSDF volume feasible on current CPU-based architectures, too [107]. An advantage of using octrees is not only that they are memory efficient but they allow geometry intersection testing to be done much faster than on a regular grid. Octrees have been extensively used in computer graphics as an acceleration structure for ray-tracing, path-tracing and collision detection, to name a few applications. Although octrees may be considered the industry standard representation for volumetric space partitioning, they are better suited for applications where the data structure is built once and used many times. It is also worth noting that the memory saved by partitioning the space into an octree is *variable*, and depends on the geometry to a great extent.

5.1.3 Hashing

Although efficient space-partitioning schemes can reduce the memory requirements of TSDFs drastically, there is still some redundancy in the representation. Because the truncated, empty regions are still represented (although at a coarser granularity) and because the higher levels of the hierarchy generally contain a summarized representation of their lower levels, there is still some redundant information being kept on multiple levels. An even more compact representation can be made by using a hash function $H(x, y, z)$ that maps world coordinates to voxels [108]. Although some memory is required to store the hash table itself, the overall memory use is typically around 1:16 of a similar regular grid (compared to twice of that for octrees). An advantage of using a hash table is that it doesn't a priori impose bounds on the volume to be reconstructed and by design grows the map as needed.

5.1.4 Moving Volumes

Although the cited methods increase the size of the feasible TSDF volume, the main motivation behind using the TSDF may be of instrumental nature: e.g. to enable robust and efficient pose estimation or to give de-noised representation of a workspace for collision-free grasp planning and object recognition [109]. In such cases, it may not be necessary, or even desirable to maintain a large-scale map, and the TSDF may thus only be required to exist around the space currently occupied by the robot. This insight has led to approaches that circumvent the bounds imposed by a fixed volume size by translating the active volume [110] and optionally also storing the cells that exit the boundary of the TSDF as a triangulated surface [111]. Translating the TSDF volume allows the robot to move further while locally enjoying the benefits of having a TSDF representation at its disposal at a fixed memory footprint. A desirable feature of a moving volume system is the ability to re-integrate the data back into the TSDF representation when one returns to a previously mapped location. While it is possible to estimate a TSDF from a given surface boundary with known orientation using the fast marching method [112] or fast sweeping method [113], the problem is difficult to solve correctly and efficiently [19] in general, due to the non-manifold nature of reconstructed meshes.

5.1.5 Dictionary Learning

Going back to the concept of compression but instead of using e.g. wavelets for sparsely decomposing the volume we can apply a more compact and representative set of basis functions. These can be learned from statistics over samples of previously reconstructed maps. This is the main idea behind sparse coded surface models [114] which use K-SVD [115] (a linear projection method) to reduce the dimensionality of textured surface patches, represented as RGB + D pixels. More

closely related to our work is the volumetric active patch model [116] which attempt to learn representative patches¹ of sampled TSDF reconstructions and then linearly combine them with estimated rotations and translations to form a reconstruction. This yields a very compact representation since a single code word suffices for representing a given shape in all of its possible orientations. An active patch model obtains robustness to viewpoint variations at the expense of requiring an iterative method for jointly optimizing pose *and* shape parameters at run-time.

5.2 Unsupervised learning for TSDF compression

In this section we will address the issue of TSDF compression by an alternative strategy. Here we investigate encoding (and subsequently decoding) the TSDF in a low-dimensional feature space by projection onto a learned set of basis (eigen-) vectors derived via principal component analysis [117] (PCA) of a large data set of sample reconstructions. We shall also see that this compression method preserves important structures in the data while filtering out noise, allowing for more stable camera-tracking against the model, using the tracking algorithm from Chapter 3. We will see that this method compares favorably to nonlinear methods based on auto-encoders (AE) in terms of compression, but slightly less so in terms of tracking performance.

The proposed compression strategies can be applied to scenarios in which robotic agents with limited on-board memory and computational resources need to map large spaces. The actively reconstructed TSDF need only be a small fraction of the size of the entire environment and be allowed to move, as described in the related work section on “Moving Volumes”. The regions that are no longer observed can be efficiently compressed into a low-dimensional feature space and re-observed regions can be decompressed back into the original TSDF representation.

Although the compression is lossy, most of the content is lost in the higher-frequency domain which we show to even have positive side effects in terms of noise removal. When a robot re-observes previously explored regions of a compressed map, the low-dimensional feature representation may serve as a descriptor, providing an opportunity for the robot to, still in the descriptor-space, make the decision to selectively decompress regions of the map that may be of particular interest. A proof of concept for this scenario and tracking performance evaluations on lossily reconstructed maps are presented in Section 5.2.4.

5.2.1 Principal Component Analysis (PCA)

PCA [117] is a method for obtaining a linear transformation into a new orthogonal coordinate system. In this system, the first dimension is associated with the

¹it is a bit misleading to call them patches in 3D, as they are essentially voxel blocks

direction, in a data set, that exhibits the largest variance. The second dimension is aligned with a direction, perpendicular to the first, along which the second most variance is exhibited and so on. We achieve this by first subtracting the data set mean α_0 from each sample in the data matrix (centering) and then applying a singular value decomposition (SVD).

Let each sample reconstruction in the data-set, which we will elaborate upon further in Section 5.2.3, be a block of $16 \times 16 \times 16$ voxels (unrolled into a vector of 4096 elements). From an entire set of such samples, SVD produces a principal component matrix of 4096×4096 entries. Multiplying a (centered) vector from the input space, i.e., a voxel block unrolled into a vector (of 4096 elements) with this matrix results in a mapping from input space into a new space of orthogonal basis vectors, one for each singular value. The magnitude of these singular values relates to the amount of variance observed along their associated basis vector in the data set and can be sorted in descending order, the smallest of which are typically close to zero. This suggests that some columns from the principal component matrix can be removed, as they are related to the lower-variance components.

Now, multiplying a (centered) vector from the **input space**, with a *reduced* set of principal components then projects the voxel data onto a *lower*-dimensional space. Inversely, multiplying a lower-dimensional vector from the **feature space** with the transpose of the principal component matrix, maps back into the input space. The resulting voxel block is then a low-rank approximation of the initial data.

Writing the singular value decomposition of our *centered* data-matrix Y_0 as

$$[U, \Sigma, V] = \text{svd}(Y_0). \quad (5.1)$$

The matrix $A = U\Sigma$ contain the *principal components* of the data. For any given sample, $p \in \mathbb{R}^{4096}$ in input space, we can generate a descriptor d_p :

$$d_p = A^T(p - \alpha_0). \quad (5.2)$$

Mapping back to the input space is done by computing,

$$\hat{p} = \alpha_0 + A d_p. \quad (5.3)$$

By removing columns from A , we can thus use PCA to design low-dimensional descriptors for the high-dimensional voxel chunks, while making use of statistics related to the training data set to pick a good approximation.

5.2.2 Artificial Neural Network

Artificial neural networks (ANNs) are often described using biological analogies, as a nonlinear computational architecture with several layers that connect multiple inputs to outputs as the weighted sum of their contributions, plus some

bias. For a fully connected neural network, this is essentially a vector-matrix product followed by addition. Described is an affine function that could be replaced by a single layer of multiplication without any loss. However, nonlinear properties are introduced by applying an element-wise *activation function* on the vector resulting from the computation at each layer. A descriptor d_p can thus be computed from the input vector p by chaining the multiplications with layers L_1 through L_k interleaved with element-wise activation functions f ,

$$d_p = f(L_k \cdots f(L_2 f(L_1 p + b_1) + b_2) \cdots + b_k). \quad (5.4)$$

Finding an appropriate set of weights for the layers $L_{1\dots k}$ and biases $b_{1\dots k}$ is done by comparing the output of the network with some reference and minimizing an objective related to this error using numerical optimization techniques. This optimization process is appropriately called “training” the network.

Training an ANN as an auto-encoder can be done in a straightforward manner by setting its desired output to be equal to its input and employing e.g., back-propagation [118] to minimize the difference over a large quantity of representative data from the target domain. For some form of encoding to occur, it is required that somewhere in between the input layer and output layer, there exists an intermediary hidden layer whose dimensionality is smaller than those of the input and output (which are of the same size). We refer to this intermediate “bottleneck” layer as a code or feature layer. The portion of the ANN up until the feature layer can then be treated as an encoder and the portion after is treated as a decoder. For practical reasons (particularly when layer-wise unsupervised pre-training is involved [119]), it makes sense to keep the encoder and decoder symmetric.

5.2.3 Methodology

Training Data

The data set used for training consists of sub-volumes sampled from 3D reconstructions of real-world industrial and office environments c.f. Figure 5.5. These reconstructions are obtained by fusing sequences of depth images from an Asus Xtion Pro camera into a TSDF as described in Chapter 2 with truncation limits set to $d_{\min} = -0.04$ and $d_{\max} = 0.1$. Camera pose estimates were produced by the algorithm described in Chapter 3 (though any method with low drift would do just as well). The real-world data are augmented by synthetic samples of TSDFs, procedurally generated using **libsdf**², an open-source C++ library that implements simple implicit geometric primitives (as described in [24, 120]). Some examples from our synthetic data set can be seen in Figure 5.6

Sub-volumes were sampled from the real-world reconstructions by taking $16 \times 16 \times 16$ samples at every eight voxels along each spatial dimension and

²<https://github.com/dcanalhas/libsdf>

permuting the indexing order along each axis for every sample to generate five additional reflections at each location. Distance values were then mapped from the interval $[d_{\min}, d_{\max}]$ to $[0, 1]$ and saved. Furthermore, to avoid a disproportionate amount of samples of empty space, sub-volumes for which the mean sum of normalized ($\in [0, 1]$) distances is below 0.95 were discarded.

The procedurally generated shapes were sampled in the same manner as the real-world data. It was generated by randomly picking a sequence of randomly parametrized and oriented shapes from several shape categories representing fragments of: flat surfaces, concave edges, convex edges, concave corners, convex corners, rounded surfaces, and rounded edges. For non-oriented surfaces, convexity and concavity does not matter, since one is representable by a rotation of the other, but need to be represented as separate cases here, since the sign is inverted in the TSDF. The distances are truncated and converted to the same intervals as with the real data. The use of synthetic data allows generating training examples in a vast number of poses, with a greater degree of geometric variation than would be feasible to collect manually through scene reconstructions alone.

The sub-volumes, when unrolled, define an input of a dimensionality equal to $n = 4096$ (i.e., 16^3), and the combined number of samples $m = 200,000$. Our data set is then $\mathbf{X} \in \{\mathbb{R}^{m \times n} | 0 \leq x_{i,j} \leq 1\}$.

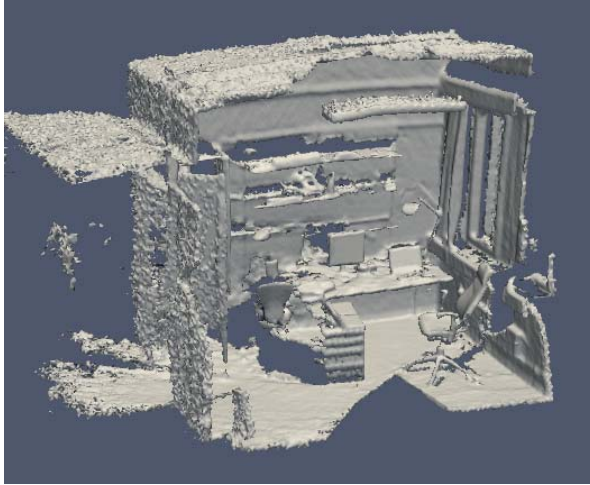


Figure 5.5: Example from the real-world data, showing the extracted zero level set as a polygonal surface mesh. The picture depicts a partial reconstruction of a small office environment. A sliding volume sampling strategy, with multiple reorderings of the voxel indexing was used to extract a large number of real-world sample reconstructions for the training data.

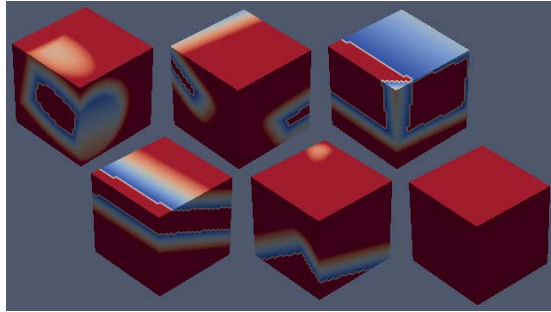


Figure 5.6: Examples from the synthetic data set showing a variety of shapes represented by truncated distance fields, sampled onto a small volume containing 4096 voxels.

Evaluation Methodology

Having defined methods for obtaining features, either using PCA or ANN, what is a good balance between compression ratio and map quality in the context of robotics? We explore this question by means of two different fitness quality measures: reconstruction fidelity and ego-motion estimation performance. To aid in our analysis, we use a publicly available RGB-D data set [59] with ground truth pose estimates provided by an independent external camera-tracking system. Using the provided ground truth poses, we generate a map, by fusing the depth images into a TSDF representation. This produces a ground truth map. We chose *teddy*, *room*, *desk*, *desk2*, *360* and *plant* from the *freiburg-1* collection for evaluation as these are representative of real-world challenges that arise in simultaneous localization and mapping (SLAM) and visual odometry, including motion blur, sensor noise and occasional lack of geometric structure needed for tracking. We do not use the RGB components of the data for any purpose in this work. We perform two types of experiments, designed to test the reconstruction fidelity and the reliability of tracking algorithms operating on the reconstructed volume.

As a measure for reconstruction error, we compute the mean squared errors of the decoded distance fields relative to the input. This metric is relevant to path planning, manipulation and object detection tasks since it indirectly relates to the fidelity of surface locations. For each data set, using each encoder/decoder, we compute a lossy version of the original data and report the average and standard deviation across all data sets.

Ego-motion estimation performance is measured by the absolute trajectory error (ATE) [59]. The absolute trajectory error is the integrated distance between all pose estimates relative to the ground truth trajectory. The evaluations are performed by loading a complete TSDF map into memory and setting the initial pose according to ground truth. Then, as depth images are loaded from the

RGB-D data set, we estimate the camera transformation that minimizes the point to model distance for each new frame. The evaluation was performed on all the data sets, processed through each compression and subsequent decompression method. As baselines, we also included the original map, and the map processed with a Gaussian blur kernel of size $9 \times 9 \times 9$ voxels and a σ parameter of $4/3$ voxels.

Building on the experimental results, we are able to demonstrate two applications: selective map decompression based on descriptor matching and large-scale mapping in a dense TSDF space, by fast on-the-fly compression and decompression.

5.2.4 Experimental Results

The PCA basis was produced, using the dimensionality reduction tools from the **scikit-learn** [121] library. Autoencoders were trained using **pylearn2** [122] using batch gradient descent with the change in reconstruction error on a validation data set as a stopping criterion. The data set was split into 400 batches containing 500 samples each, of which 300 batches were used for training, 50 for testing, and 50 for validation. The networks use *sigmoid* activation units and contain, in each subsequent layer, 4096, 512, d , 512, 4096 nodes respectively, with d representing the number of dimensions of the descriptor. The layers up until d are used for encoding, and the layers after d are used for decoding.

The runtime implementation for the encoder/decoder architectures was done using the cuBLAS [123] and Thrust [124] libraries, enabling matrix-vector and array computation on the GPU.

Reconstruction Error

We report the average reconstruction error over all non-empty blocks in all data sets and the standard deviation among data sets in Table 5.1. The reconstruction errors obtained strongly suggest that increasing the size of the codes for individual encoders yields better performance, though with diminishing returns. Several attempts were made, to out-perform the PCA approach, using Artificial Neural Networks (ANN) trained as auto-encoders, but this was generally unsuccessful. PCA-based encoders, using 32, 64 and 128 components, produce better results than ANN encoders in all of our experiments

Reconstruction Method	Reconstruction Error (MSE) $\pm \sigma$	Mean ATE (m) $\pm \sigma$	Median ATE (m)
Original data	-	0.63 ± 0.56	0.72
PCA 32	42.8 ± 6.39	0.23 ± 0.34	0.049
PCA 64	33.81 ± 3.88	0.36 ± 0.47	0.047
PCA 128	26.88 ± 2.57	0.56 ± 0.68	0.12
NN 32	59.66 ± 7.17	0.16 ± 0.29	0.057
NN 64	49.18 ± 4.72	0.12 ± 0.17	0.049
NN 128	45.66 ± 4.25	0.13 ± 0.20	0.046
Gaussian Blur $9 \times 9 \times 9$	-	0.10 ± 0.17	0.038

Table 5.1: Average reconstruction and ego-motion estimation results across all data sets. PCA - Principal component analysis compression, NN - Artificial neural network compression, MSE - Mean squared error, ATE - Absolute trajectory error. The suffixes: 32, 64 and 128 correspond to compression ratios of 128:1, 64:1 and 32:1, respectively. Empty cells are discarded, and do not contribute to the reconstruction error.

The best overall reconstruction performance is given by the PCA encoder/decoder, using 128 components. We illustrate this with an image from the *teddy* data set in Figure 5.7. Note that the decoded data set is smoother, so, in a sense, the measured discrepancy is partly related to a qualitative improvement.

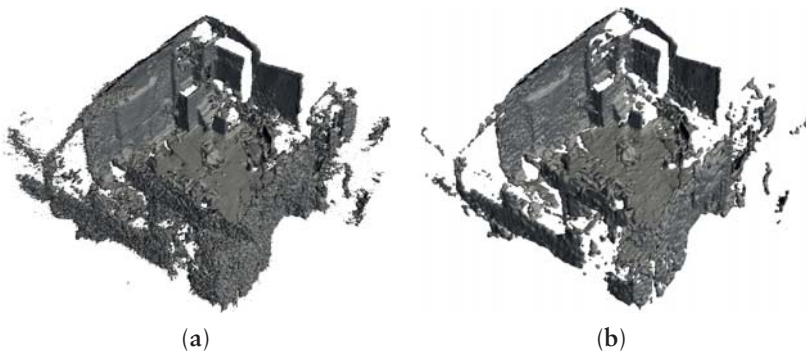


Figure 5.7: Example reconstruction using a PCA basis with 128 components. The reconstructed version (b) includes some blocking artifacts, visible as tiles on the floor of the room, but contains visibly less noise than the original map (a).

Ego-Motion Estimation

The ego-motion estimation, performed by the SDF Tracker algorithm, described in Chapter 3 uses the TSDF as a cost function to which subsequent 3D points are aligned. This requires that the gradient of the TSDF be of correct magnitude and point in the right direction. To get a good alignment, the minimum absolute distance should coincide with the actual location of the surface.

In spite of being given challenging camera trajectories, performance using the decoded maps is on average better than with the unaltered map. However, tracking may fail for various reasons, e.g., when there is little overlap between successive frames, when the model or depth image contains noise or when there is not enough geometric variation to properly constrain the pose estimation. In some of these cases, the maps that offer simplified approximations to the original distance field fare better. The robustness in tracking is most likely owed to the denoising effect that the encoding has, as evidenced by the performance on the Gaussian blurred map. Of the encoded maps, we see that the auto-encoder compression results in better pose estimation. In Figure 5.8, we see a slice through a volume color-coded by distance.

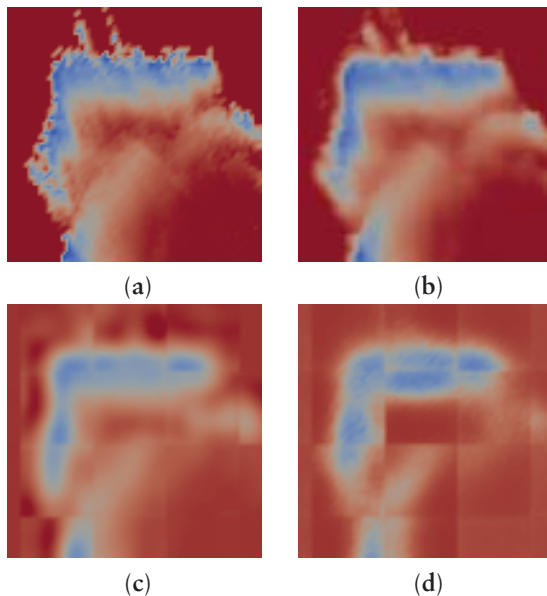


Figure 5.8: A slice through the distance field reconstructed through different methods, using 64-element encodings. Shown here are (a) the original map; (b) the Gaussian filtered map; (c) PCA reconstruction; and (d) auto-encoder reconstruction.

Here, we note that, even though the PCA-based map is more similar to the original than the auto-encoder reconstruction, on the left side of the image, it is evident that the field is not monotonically increasing away from the surface. Such artifacts cause the field gradient to point in the wrong direction, possibly contributing to failure in finding the correct alignment. The large difference between the median and mean values for the pose estimation errors is indicative of mostly accurate pose estimations, with occasional gross misalignments.

Selective Feature-Based Map Expansion

Although the descriptors we obtain are clearly not invariant to affine transformations (if they were, the decompression could not reproduce the field in its correct location/orientation), we can still create descriptor-based models for geometries of particular interest by sampling their TSDFs over the range of transformations to which we want the model to be invariant. If information about the orientation of the map is known a priori, e.g., some dominant structures are axis-aligned with the voxel lattice, or dominant structures are orthogonal to each other, the models can be made even smaller. In the example illustrated in Figure 5.9, a descriptor-based model for floors was first created by encoding the TSDFs of horizontal planes at 15 different offsets, generating one 64-element vector each. Each descriptor in the compressed map can then be compared to this small model by the squared norm of their differences and only those beneath a threshold of similarity need to be considered for expansion. Here, an advantage of the PCA-based encoding becomes evident: since PCA generates its linear subspace in an ordered manner, feature vectors of different dimensionality can be tested for similarity up to the number of elements of the smallest, i.e., a 32-dimensional feature descriptor can be matched against the first half of a 64-dimensional feature descriptor. This property is useful in handling multiple levels of compression, for different applications, whilst maintaining a common way to describe them.

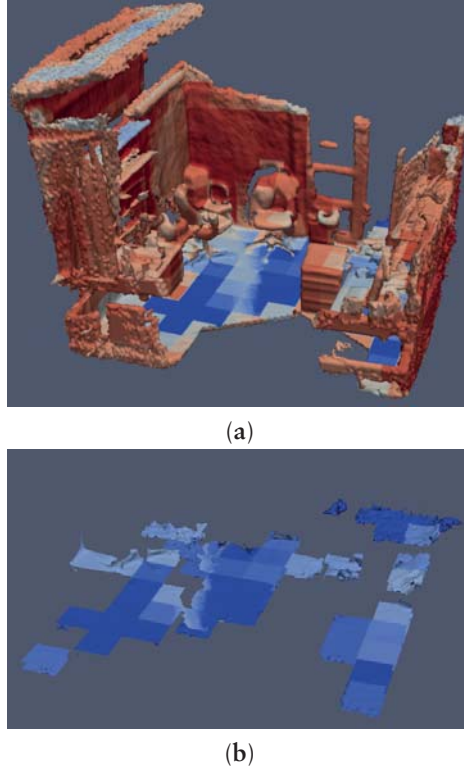


Figure 5.9: Selective reconstruction of floor surfaces. Given a compressed map, the minimum distance for each compressed block, to a set of descriptors that relate to horizontal planes can be computed (e.g., floors). Only the blocks that are similar enough to this set of descriptors need to be considered for actual decompression. In (a), the uncompressed map is shown, with each region colored according to its descriptor’s distance to the set of descriptors that relate to floors. In (b), we see the selectively expanded floor cells.

Large-Scale Mapping

By extending the SDF Tracker algorithm with a moving active volume centered around the camera translation vector, for every 16-voxel increment, we can encode the voxel blocks exiting the active TSDF on the lagging end, (if not empty), and decode the voxel blocks entering the active TSDF on the front end (if they contain a previously stored descriptor). This allows mapping much larger areas than without compression, since each voxel block can be reduced from its 4096 voxels to the chosen size of a descriptor or a token, if the block is empty. Since the actual descriptor computation happens on the GPU, the performance

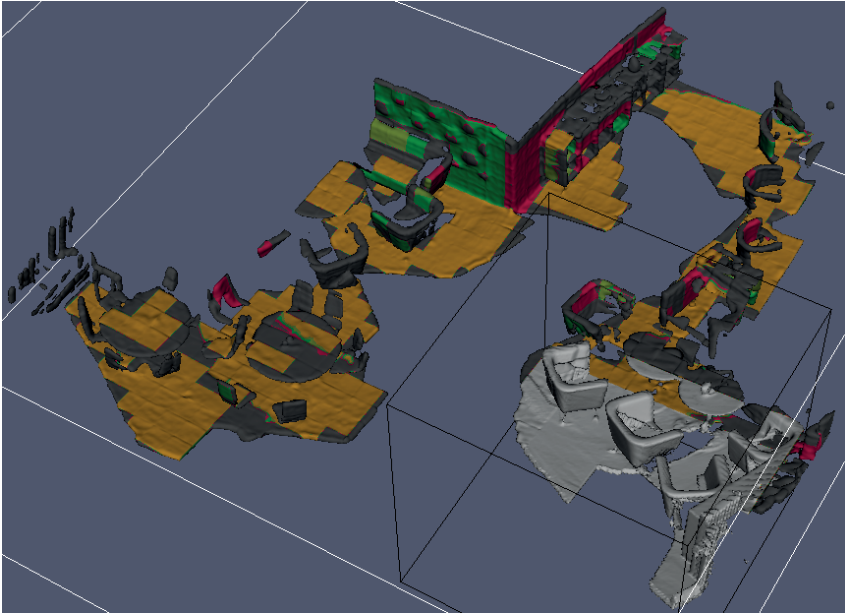


Figure 5.10: Large scale mapping, enabled by real-time encoding and decoding of data as the reconstruction volume translates to keep the camera within its bounds. The active reconstruction volume is shown in black outlines. Each block is color-coded according to which descriptors it mostly resembles. In this example we have compared the descriptors against those associated with walls (in two directions) and the floor, as described in the previous section.

impact on the tracking and mapping part of the algorithm is minimal. An example of this is shown in Figure 5.10.

The tracking and mapping algorithm itself is a real-time capable system with performance that scales with the size of the volume and resolution of the depth images given as input. While it only handles volumes of approximately 160^3 voxels at QVGA (quarter of VGA resolution: 320×240 pixels) resolution (320×240 pixels) and lower at 30 Hz, it does so solely on an Intel i7-4770K 3.50 GHz CPU. This leaves the GPU free to perform on-the-fly compression and decompression.

Timing the execution of copying data to the GPU, encoding, decoding and copying it back to the main CPU memory results in an average between 405 and 520 μ s per block of 16^3 voxels on a Nvidia GTX Titan GPU. For an active TSDF volume of $160 \times 160 \times 160$ voxels, shifting one voxel-block sideways would result in 100 blocks needing to be encoded, 100 blocks decoded, and memory in

both directions’ transfers, in the worst case. Our round-trip execution timing puts this operation at approximately 41 to 52 milliseconds.

The span in timings depend on the encoding method used, with PCA-based encoding representing the lower end and ANN the upper, for descriptors of 128 elements. A reference implementation is made available ³.

5.3 Discussion

In this section, we presented dimensionality reduction of TSDF volumes. We proposed PCA and ANN encoding strategies and evaluated their performance with respect to a camera tracking application and to reconstruction error. We demonstrated that we can compress volumetric data using PCA and neural networks to small sizes (between 128:1 and 32:1) and still use them in camera tracking applications with good results. We showed that PCA produces superior reconstruction results and although auto-encoders have inherently greater expressive power, training them is not straightforward, often resulting in lower quality reconstructions. Neural networks offered slightly better performance in ego-motion estimation applications, however. The discrepancy between reconstruction error and tracking performance is likely to be the result of smoothing, supported by good performance observed on maps that were simply subjected to a low-pass filter. Finally, we have shown that this class of methods can be successfully applied to both compress and imbue the data with some low-level semantic meaning and suggested an application in which both of these characteristics are simultaneously desirable.

From the results presented in this section, it is clear that the proposed feature encodings are not invariant to rigid-body transformations. Therefore, we should not expect shapes of similar objects with different orientations to cluster in the lower-dimensional space, as this was not part of the objective function used to generate the mapping. An efficient method for obtaining a low-dimensional representation as well as a reliable transformation into some canonical frame of reference would pave the way for many interesting applications in semantic mapping and scene understanding. Finding features in a canonical frame of reference, along with an estimate for the pose has been attempted with deep neural networks [125]. While the quality obtainable using *deep* neural network architectures is impressive, the number of computational layers in conjunction with the amount of voxel blocks to be processed in parallel likely results in a solution that is beyond the computational budget of a real-time system today. Furthermore, it seems unfortunate that pose-estimation ultimately has to occur in the voxel domain. Given that the transformation to the low dimensional space is a simple affine function (at least for the PCA-based encoding) it seems intuitive that one should be able to formulate and solve the pose-estimation

³https://github.com/dcanelhas/sdf_tracker-LS

problem in the reduced space with a lower memory requirement in all stages of computation. Investigating this possibility remains an interesting open problem.

Chapter 6

Minimalistic Representations from TSDF

So far we have defined the TSDF as a voxel-based representation of surfaces in 3D space and described how to estimate it based on depth image inputs with known poses (Chapter 2). In Chapter 3 we derived a method for tracking the pose of a depth camera against the model to relax the assumption of known poses. We then evaluated the use of feature detectors and descriptors in TSDFs (Chapter 4). As a side-effect of the effort on 3D descriptors, we also formulated an efficient single-pass compression algorithm. The approach developed can be used in real-time to mitigate the memory complexity of volumetric representations, whilst simultaneously providing weak semantic labeling and noise-filtering to the surface representation (Chapter 5). Although the proposed method of compression is one way of addressing the memory complexity problem, it only does so in a limited sense. The compression hides some of the redundancy by recognizing empty space and abbreviating it with a token, or by projecting chunks of data down onto a lower-dimensional space. As an analogy on the practicality of this approach, one might imagine a volume of water that was previously stored in teacups as now being kept in buckets. Granted, there are fewer containers to keep track of, but what would a solution look like, if we could think a bit outside the box (or bucket, for that matter)?

The need for representing large maps is self-evident in the case for mobile robotics if *mobility* is to be a design goal in any non-trivial sense of the word. The map representation thus needs to fit into a small memory footprint. However, it needs to strike a balance between compactness and usefulness in tasks relevant to mobile robot applications, such as place recognition. One way to achieve the stated goal is to continue along the path of local feature detection and description. However, place recognition by matching local geometric feature descriptors is expected to be a difficult task, since local shapes are sparse and highly repetitive. This statement is true in many application domains. For instance, forests tend to be filled with many tall cylinders and offices tend to contain mostly flat surfaces

at right-angles from each other with the occasional coffee-filled cylinders. In either case, knowing *which* shapes were seen is not helpful. In such cases where the geometry within an environment tends to be mostly repetitive, e.g. if there are several instances of the same objects, the uniqueness of one location compared to another may be better determined by the relative arrangement of said geometry.

Motivated by this intuition and in possession of the algorithmic tools presented so far, we will devise a novel graph representation that is both light-weight and captures the spatial interrelations between geometric features in a scene. In this graph, the nodes will simply be the output of a 3D feature detector and edges will be said to exist between pairs of nodes that can be joined by a line drawn along a surface, represented by the TSDF. We will refer to this representation as a sparse, stable scene-graph (SSSG) [126]. As a visual aid, the relationship between TSDF and SSSG is illustrated in Fig. 6.1.

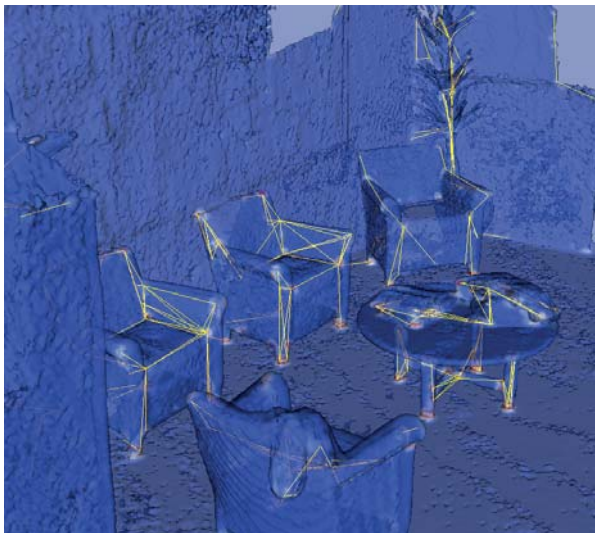


Figure 6.1: Lounge dataset reconstructed as a TSDF, visualized as its triangulated zero-set, and its associated SSSG

The proposed graph structure can be seen as related to a broader class of representations used for model-based robot vision known as *relational graphs* [127, 128]. While relational graphs typically incorporate more semantic meaning in the nodes, we will remain on a lower level of abstraction from the data, focusing on geometrically linked points of interest. Based on the methods we have developed so far, let us now see how such a graph can be constructed efficiently.

6.1 SSSG - Construction



(a) A view of the TSDF surface obtained by reconstructing the “Lounge” data-set and ray-marching the reconstruction with a diffuse shading model



(b) Computing the feature response function of the Harris corner detector for the TSDF produces spikes at the locations where geometry is curved in several directions, shown here in pink

Figure 6.2: TSDF and feature response function, shown side by side

To begin this section, let us make the assumption that we have reconstructed some scene content within the bounds of a TSDF volume. As an example, we can take a scene from Zhou’s 3D scene data-set [99], shown in Fig. 6.2(a) as a ray-marched surface interface, with simple Phong lighting. Applying the volumetric Harris corner detector and coloring the voxels according to its feature response function, we get highlights at corners and along creases and edges in the geometry. This is illustrated in Fig. 6.2(b).

Following the Harris corner detector algorithm, after performing non-maxima suppression and thresholding, we are left with a list of detected feature points. These represent all the local maxima of the feature response function that exceed a threshold value. It is not guaranteed that these features will coincide exactly with the surface. Feature detectors may in general tend to peak at off-surface locations and if discrete in nature, will output the location of a voxel rather than a real-valued coordinate. This is illustrated by the 2D example in Fig 6.3. There are at least two simple ways to work around the issue of features being off-surface, for general feature detectors:

1. In Chapter 3 we showed how one can use the gradient and magnitude of the TSDF to find the nearest surface point for a location within the non-truncated region of the field. Often, detectors require computing image gradients, so it may already be available at no extra cost.
2. Due to the surface being represented as a distance field whose magnitude increases further from it, we can modify any response function to decrease its value based on the absolute TSDF value.

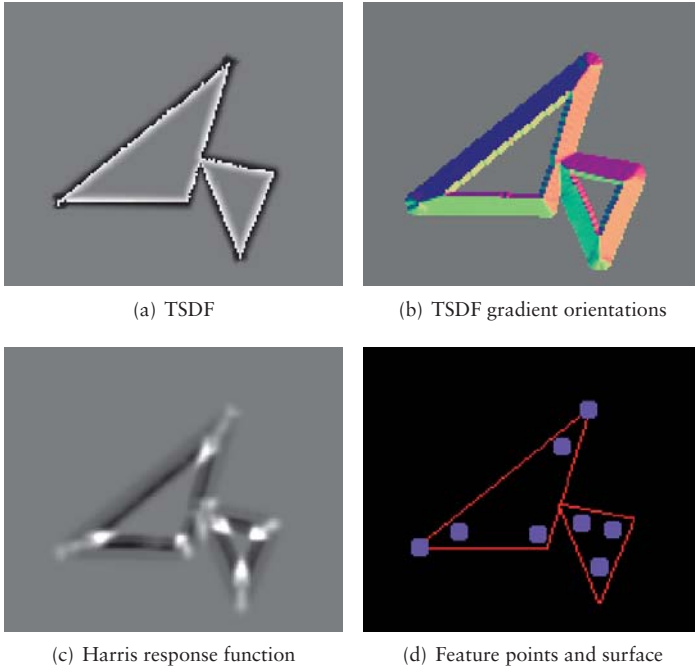


Figure 6.3: 2D example of Harris feature detection in a TSDF showing that the maximal response does not necessarily coincide with a surface location.

In the specific case of Harris corners, we observe that features *are* detected at close proximity to the surface, but that the response is much stronger at the negative truncation limit due to the strong gradient caused by the sudden transition from D_{\min} to D_{\max} . This merely serves as yet another reminder for why gradients based on truncated values should be avoided and we note that when following this advice we obtain features located near the surface boundary instead. Regarding the discrete nature of the feature detector, we have found that in practice, integer voxel coordinates tend to be close enough for reliably constructing an SSSG.

To construct the graph, we could iterate over all the node-pairs and test if the lines that join them run along a surface and if so, add them to a list of edges. A different way to do this that is better suited for parallel processing is to initially assume that the graph is fully connected, prune the edges whose existence are not supported by the data and write out the remaining connections as a final step. The graph can be expressed as a matrix that relates a particular edge (represented as an element in the matrix) to its two endpoint nodes, represented as the row

and column index. Furthermore, whether two vertices are connected or not, does not change based on the order in which they are picked, meaning that the graph is undirected. An undirected graph can be represented by a triangular matrix (omitting the main diagonal, since this would imply trivially connecting each node to itself). To give an example, consider the following fully connected undirected graph:

$$\mathbf{G} = \begin{bmatrix} - & - & - & - & \dots \\ 1 & - & - & - & \dots \\ 1 & 1 & - & - & \dots \\ 1 & 1 & 1 & - & \dots \\ 1 & 1 & 1 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad (6.1)$$

We note that the triangular structure of the graph is one that is not directly divisible into even-sized workloads. If we were to process each row in parallel, the first thread would be assigned a single edge to verify, while the last would be processing as many edges as there are nodes in the graph (i.e. columns in the matrix). To get around this problem, let us consider what happens if we number the edges sequentially:

$$\mathbf{G} = \begin{bmatrix} - & - & - & - & \dots \\ 0 & - & - & - & \dots \\ 1 & 2 & - & - & \dots \\ 3 & 4 & 5 & - & \dots \\ 6 & 7 & 8 & 9 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad (6.2)$$

We see that the first column contains the following series: 0, 1, 3, 6, 10 etc. These are known as *triangular numbers* [129] and have a well-defined pattern. Making use of the fact that every row will be indexed by a triangular number allows us to take any of the entries in the matrix as an index for an edge i_e , and determine its corresponding row and column indices directly¹. This is done by taking the triangular root of the edge index as follows:

$$\text{row} = \left\lfloor \frac{\sqrt{8i_e + 1} - 1}{2} \right\rfloor + 1 \quad (6.3)$$

$$s = \frac{\text{row}(\text{row} - 1)}{2} \quad (6.4)$$

$$\text{col} = i_e - s \quad (6.5)$$

without actually having to generate or store the matrix in memory. We may now treat the entire matrix as a single linear index of edges and can split the work

¹indices starting at zero

evenly with ease. We still require the total number of edges in the fully connected graph. For n_f nodes, the total number of edges is given by the expression: $n_e = \frac{n_f(n_f-1)}{2}$ (essentially the area of a triangle of base and height equal to n_f , and $n_f - 1$, respectively)

The next step is to prune the graph such that it only contains the edges that are embedded in the surface. To this end we launch n_e separate threads on a GPU. Each thread is given a unique thread index by the parallel programming API. We take this index as being related to a matrix entry, i.e. an edge index i_e . It is then straightforward to retrieve the feature points referenced by the row and column index using equations (6.3) and (6.5).

Testing whether two nodes are connected by a surface in the TSDF is, due to the discrete representation of the signed distances, done by sampling along the line that connects them. By linearly interpolating between endpoint feature locations we can query the TSDF at a number of points along each edge and reject the edge based on some statistic about the measurements made along it. For an arbitrary edge rejection threshold we can compare it to any of the following statistics, yielding different results:

- Maximum absolute value – this forces the entire edge to lie within a pair of TSDF level-sets around zero. It is the most restrictive option in terms of what will be accepted as a valid edge, and connections tend to break for both noisy surfaces and small occluded regions that may not have been reconstructed,
- Mean absolute value – the mean is more robust to disturbances than the maximum. Edges can with greater ease be retained in the presence of noise and holes in the reconstructed surface. In situations where noise is not an issue and the main problem is connecting edges across holes in the surface, due to the mean’s sensitivity to outliers, the rejection threshold may need to be set higher. However, a higher threshold may sometimes cause straight edges to also be connected through curved surfaces.
- Percentiles – An adequate percentile allows us to set a low threshold, but still bridge gaps in the surface that occur between the endpoints, ensuring that edges are connected via flat surfaces. Higher percentiles such as e.g. the median, may cause up to half of the edge to extend out into free space from an otherwise continuous, flat surface, to produce a link with a seemingly disconnected feature point. In other words, the edge-pruning may spend all of its gap-bridging potential on a single section of the edge.

The number of points along the edge to test can be made dependent on the length of the edge or constant. A constant number of steps makes the spacing denser for short edges. Regardless, it may be wise to keep the number of steps constant, to avoid variations in the amount of work per thread. In parallel execution models such as in GPU computing, the remaining threads within a thread group will typically wait idly for the last one to terminate.

We may also choose to reject edges whose endpoints are either too far apart (to promote local clusters of object-sized subgraphs), or too close (to avoid representing small-scale clutter). The minimum distance between feature points will already be determined by the non-maxima suppression step of the feature detection, but this offers an additional way of influencing the minimum edge-length. We store pass or fail decision in a binary vector of the same size as the number of edges and a standard stream compaction [124] operation is then used to extract the pruned graph.

Additionally, we may wish to remove all the disconnected feature points from the graph. Computing a histogram² of the edge list, using as many bins as there are features, produces a count for how many times each feature is referenced in the graph. Another stream compaction operation then removes the features whose reference count is lower than one (or some other threshold, if one has a reason to prefer more densely connected graphs). In summary, please refer to Algorithm 3

²A task likewise suitable for the GPU, see <https://github.com/thrust/thrust/blob/master/examples/histogram.cu> for an example implementation in cuda

Algorithm 3 Parallel SSSG construction and deletion of unreferenced nodes in the graph. The function `isConnectedTo()` implements sampling of the TSDF at an arbitrary number of steps between the points p and q using, though not limited to, any of the edge rejection criteria suggested in the text. The function `remapIndices()` requires performing an *exclusive scan* operation on the histogram to give a cumulative count of the occurrence of zeros. This count indicates precisely what the references in the edge list will have to be remapped to, once the unreferenced nodes in the graph have been removed. The required operations, as well as the expressions `copyIf()` and `histogram()` are subroutines for which parallel implementations have been made widely available [124].

Input: `feature_list`, TSDF

Output: `edge_list`, `reduced_feature_list`

```

1: {Number of features}
    $n_f \leftarrow |\text{feature\_list}|$ 
2: {Number of edges}
    $n_e \leftarrow \frac{n_f(n_f-1)}{2}$ 
3: resize edge_list to  $n_e$ 
4: edge_mask  $\leftarrow$  zeros( $n_e$ )
5: for  $i_e \in 0 : n_e$ , in parallel do
6:    $\text{row} \leftarrow \lfloor \frac{\sqrt{8i_e+1}-1}{2} \rfloor + 1$ 
7:    $s \leftarrow \frac{\text{row}(\text{row}-1)}{2}$ 
8:    $\text{col} \leftarrow i_e - s$ 
9:    $p \leftarrow \text{feature\_list}[\text{row}]$ 
10:   $q \leftarrow \text{feature\_list}[\text{col}]$ 
11:  edge_list[ $i_e$ ]  $\leftarrow$  {row, col}
12:  if isConnectedTo( $p, q, \text{TSDF}$ ) then
13:    edge_mask[ $i_e$ ]  $\leftarrow$  true
14:  else
15:    edge_mask[ $i_e$ ]  $\leftarrow$  false
16: edge_list  $\leftarrow$  copyIf(edge_list, edge_mask)
17: {histogram of  $n_f$  bins}
    $h_e \leftarrow \text{histogram}(\text{edge\_list}, n_f)$ 
18: {deletes unreferenced nodes from the graph}
   reduced_feature_list  $\leftarrow$  copyIf(feature_list,  $h_e > 0$ )
19: edge_list  $\leftarrow$  remapIndices(edge_list,  $h_e$ )
20: return

```

6.2 Geometric Place Recognition Using SSSG

With both the concepts and implementation details of SSSG construction laid bare, we will now turn our focus to an application within the field of robotics: Place recognition. Specifically, we are interested in knowing if matching global

geometric structure is a sufficient condition for achieving a robust place recognition system.

The place-recognition system that we will discuss here may serve as a modular extension to the large-scale mapping system that we discussed in Ch. 5. The ingredients: TSDF integration of depth data, pose estimation, and shifting working volume all remain the same. However, instead of³ compressing the voxels that exit the volume using a projection onto a lower-dimensional space, we build an SSSG of the current working volume and save it together with its associated pose estimate. Since the SSSG encodes the main geometric structures of objects and their spatial organization within the scene, we propose that robustly comparing the graphs is enough to give a robot a useful answer to the question “Have I been here before?”

The place-recognition back-end needs to compare the current SSSG to all the previous graphs. The comparisons themselves are performed using my GPU implementation⁴ of the RANSAC [130] algorithm.

6.2.1 Related work - 3D-NDT Histogram Matching

To put the results in the following sections into context, we will also discuss a related work, based on the three-dimensional normal distributions transform [131] (3D-NDT). The 3D-NDT, like the TSDF, is a voxel-based map representation. However, unlike in the TSDF, voxels in a 3D-NDT representation contain a 3D Gaussian distribution, fitted to the point samples collected inside the voxel. Each 3D-NDT voxel thus stores a vector representing the mean $\mu \in \mathbb{R}^3$ and a covariance matrix $\Sigma \in \mathbb{R}^{3 \times 3}$, $\Sigma = \Sigma^T$ whose eigenvectors and associated eigenvalues indicate the directions and magnitude, respectively, of the variance exhibited by the point samples for each given voxel. A place recognition system for scenes represented as 3D-NDT maps has been proposed [132] by computing 3D-NDT histograms [133].

A 3D-NDT histogram is a global descriptor that summarizes information about an entire scene, and under this interpretation it is loosely related to the concept of SSSGs. The histogram is computed by first classifying all 3D-NDT voxels in a scene as representing either spherical, linear or flat distributions, based on the relative magnitudes of the eigenvalues. The frequency of occurrence for the different classes are then binned into a histogram. The spherical and linear classes are given one bin each while the flat distributions are binned according to the direction of the underlying surface normal vector (estimated as the eigenvector associated with the smallest eigenvalue). To produce a linear index for the histogram bins that represents a discretization of all possible orientations, a spherical Fibonacci [134, 135] parametrization is used. Separate histograms are computed for voxels at three different ranges (relative to the

³or in addition to, if computational resources so permit

⁴<https://github.com/dcanelhas/cuRANSAC>

sensor origin) and concatenated to form the final descriptor. The descriptors are matched by first aligning the orientation components and then computing the Euclidean distance between the feature vectors.

6.2.2 Random Sample Consensus Matching of SSSG

Matching two graphs based on their geometry is not a problem we can generally tackle with algorithms such as ICP, since data-association by the closest point may result in a poor final alignment and high residual even for similar scenes, due to potentially large relative pose differences between the graphs. General graph-matching algorithms are designed to compute the similarity of graphs based on topological equivalence. The location of nodes and length of edges is not necessarily of importance in such cases but may be relevant in a different scenario, e.g. if we were also accounting for deformations.

In order to robustly identify possible matches between different SSSGs, we base our approach on the RANSAC algorithm. The RANSAC algorithm is a general method for parameter estimation that works well in cases where generating a hypothesis from a subset of the data and verifying it can be done quickly. It has numerous applications in robot/computer vision but also to more general curve-fitting. When applied to searching for rigid-body 3D transformations that align two models (source and target) it works by executing the following steps:

1. Randomly select a minimal set of samples from the source and target models. In 3D the required number of samples is 3. This set defines a triangular face, provided that the points are distinct. The point sets need not consist of corresponding points in the sense that they are related to the same physical locations in the source and target models in order to produce a good alignment. If the transformation that relates the two triangular faces also aligns the models, we obtain useful results, regardless. The requirement that one must randomly succeed in picking correspondences is a common misconception for how the algorithm works, and offers a slightly pessimistic outlook on the probability of finding a match. An illustration of a few selected matchings of two graphs is shown in Fig. 6.4.
2. Compute the optimal transformation that brings these minimal source and target sets into alignment. Since both sets of points are coplanar, as is the case when there are only three of them, we can use the simplified version of Horn's formula [136].
3. Apply the transformation to the entire source model;
4. Score the alignment between source and target models.

The last step requires some extra care, since it largely determines the quality of the outcome. Here, we must choose which properties of the graph to compare. For example, we can score a matching by the distances between the closest

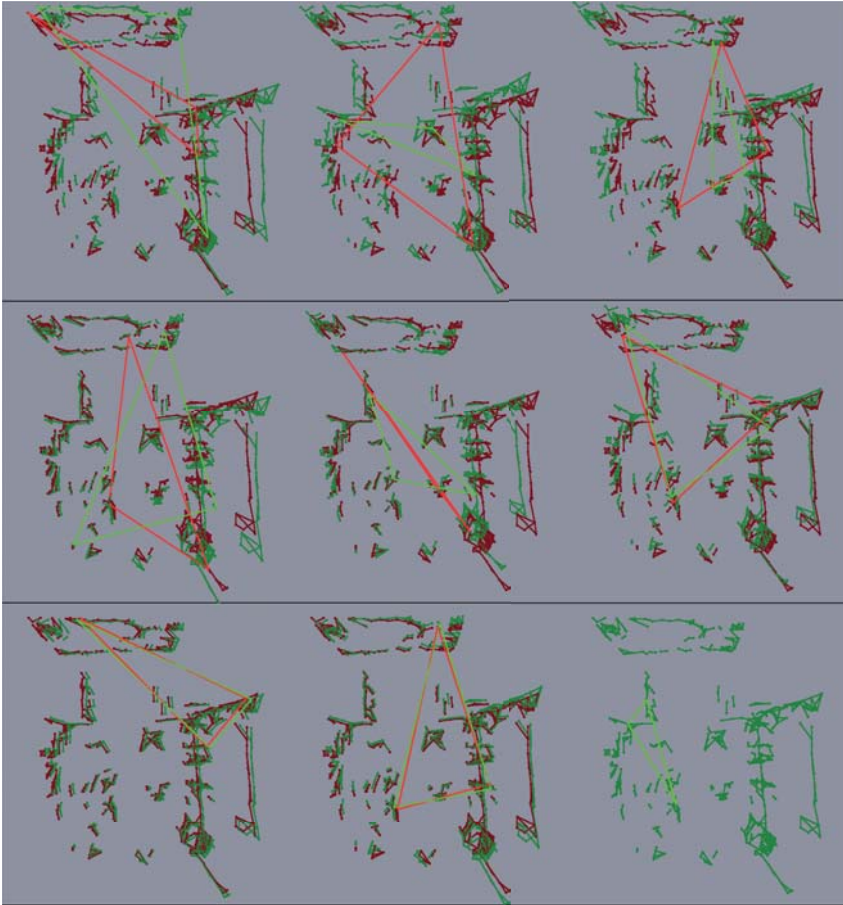


Figure 6.4: Transformations between identical source (green) and target (red) SSSGs produced by RANSAC. The brighter triangles illustrate the minimal subset of vertices selected from each to compute the current transformation. We note that the subsets are sometimes near-correspondences but often not at all. The last image is the result when three exact correspondences were selected by chance.

vertices in the source graph relative to the target graph. However this does not take edges into account, which tend to be more numerous and informative than only vertices. Another option is to take the midpoints along edges and compute the relative distances between these. One may wish to use both of these metrics in combination.

Additional properties may also be used, such as differences in how many edges are connected to each vertex, color, and edge and/or vertex labels. These different dimensions are not trivially included into a single fitness score since their quantities do not express similar physical concepts and are thus not directly comparable. We will assume that scores are computed based on the distance between edge midpoints, for the remainder of this chapter.

Given a pair of candidate matching edge midpoints, we need to make several additional choices:

- what is the largest individual sample error (i.e., Euclidean distance between candidate matches) we will admit, before discarding it as an outlier?
- what is the minimum ratio of inliers we require to accept a solution as plausible?
- what is the minimum mean error we require of the remaining inliers to accept a solution?

If the source and target models are the same, these questions are not really too interesting since none of the points can be considered outliers and we would like every point to contribute to the error, since the lowest will be associated with the correct alignment. However, when we *do* have outliers and there is only partial overlap between source and target models, we may well find transformations that simply maximize overlap get better scores than those that actually match the parts that the models have in common. Since a place recognition should be able to recognize locations based on partial observations of previously visited spaces, we have to treat the non-overlapping sections of the graph as outliers.

Choices for the thresholds then become conditioned on other aspects of the system. The individual sample error is related to the repeatability of the SSSG at a given location. If we are confident that the same features will be detected and that the same edges will be created between them, we may set this threshold fairly low, since we would then expect a correct alignment to place all the corresponding features in close proximity. If we expect the vertices and edges to merely cluster in similar locations, but to be arranged in a much less deterministic manner, either due to very generous edge-linking policies, noisy sensors or unstable feature detection, we may wish to set the individual sample error threshold higher.

The inlier ratio should correspond to the amount of expected overlap between graphs, given the above criterion. If a robot has a sensor with very wide field of view (or when the TSDF volumes that underpin the graph are initialized with large overlap, and scanned exhaustively with the sensor), we may be confident in requiring a larger ratio of inliers, since we expect to reconstruct much of the same geometry when the robot finds itself revisiting previously explored locations. On the other hand, if one is merely looking for a specific object-related subgraph within a larger scene-related graph, we may need to be much more permissive with respect to outliers.

Finally, we should set some threshold to indicate whether we accept the result of matching two given graphs as an indication of being in the same place. Informing this decision is the main contribution of this section.

6.2.3 Methodology

We will evaluate the place recognition system in two ways, firstly as a binary diagnostic test that returns positive when two graphs are deemed to be of the same location and negative, otherwise. The second evaluation will assess the error in estimated transformations between the graphs, relative to ground-truth (as estimated by a global optimization algorithm [99]) pose estimation.

Place Recognition

To evaluate the place recognition system, we use a subset of the publicly available Stanford 3D Scene data. This data-set has ground-truth camera poses obtained by global optimization, taking into account all the data (including color) in an anachronistic fashion. This data-set is characterized by longer trajectories than those of the TUM RGB-D data-set, used in Chapter 3. The trajectories also tend to form one or more loops. The sequences are shown in Table 6.1. A moving working volume is initialized around the initial pose of the camera and the depth data is fused into a TSDF. Whenever the translation component of the camera pose exceeds a threshold, we run a Harris feature detector on the current live volume and produce an SSSG that links the features through the reconstructed surfaces. To enable real-time operation, the features are detected on a point cloud representation of the TSDF, extracted at a density of one point per voxel. The SSSG, along with the current camera pose, is sent to the back-end place recognition system that compares it to all previously extracted graphs. A positive test indicates that the RANSAC error that resulted from the comparison of two graphs was below a threshold, otherwise the test is negative.

The parameters used for the place recognition system are summarized in Table 6.2. One remaining detail that ought to be mentioned is that each transformation hypothesis is tested twice. Specifically, for each edge midpoint in the source graph, we look for the closest edge midpoint in the target graph, under the proposed transformation. If the distance between them exceeds the maximum sample error, it is regarded as an outlier. We then apply the inverse transformation to the target graph and compare it to the source graph in the same way. We then take the mean of the two errors as the result. If either of the comparisons resulted in too many outliers, its error is set to a very large value, guaranteeing that the mean will be above the limit. This source-to-target and target-to-source comparison is equivalent to requiring that the intersection of the graphs be close to their union, avoiding that small features, which may conceivably fit nicely in many different nooks of a much larger graph, are reported as matching with low error.


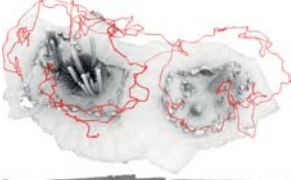

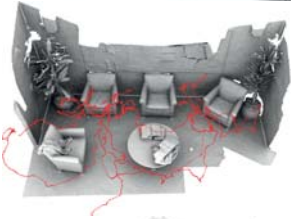
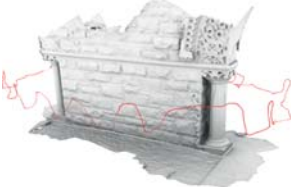
Figure	Name	Trajectory length
	Burghers of Calais	156.7m
	Cactusgarden	62.1m
	Copyroom	56.4m
	Lounge	43.1m
	Stonewall	31.2m

Table 6.1: Selection of sequences from the 3D Scenes data-set, 3D reconstruction visualized, with camera trajectory shown in red, lengths estimated by integrating pose differences in the ground-truth trajectories.

Our ground-truth label for considering a pair of graphs as the same is determined by the overlap of the surfaces, extracted as point-clouds from the TSDF reconstructions at their respective poses. If the surfaces overlap by more than 50% we consider the **condition positive**, anything less is considered negative.

number of iterations	10000
maximum sample error	0.1m
minimum inlier ratio	40%

Table 6.2: Parameters used for the RANSAC place recognition system

The surface overlap is measured by searching for the nearest neighbors from source-to-target and target-to-source point-clouds and computing the number of points, out of the total, that have a match (i.e. a neighbor within 30mm) in the other point-cloud:

$$\text{overlap} = \frac{\text{matches}_{\text{forward}} + \text{matches}_{\text{reverse}}}{|\text{points}_{\text{source}}| + |\text{points}_{\text{target}}|} \quad (6.6)$$

While surface overlap is a necessary condition for geometric place recognition, it is likely that some portions of the overlapping surfaces are rather featureless. In essence this means that in practice, some ground-truth positive conditions may be undetectable or detectable only with a high degree of uncertainty.

Pose Errors

Since RANSAC gives us not only a binary signal, but also a rigid-body transformation, we can evaluate the accuracy of these estimates, too. We can measure errors in the relative pose estimation between graphs, by comparing it to the ground truth pose differences.

6.2.4 Experimental Results

Place Recognition

The place recognition results for all data-sets can be represented visually, as done in Fig. 6.5. There we see one column for each scene in the data-set used for evaluation. The three rows in the figure correspond to (from top to bottom) computed surface overlap, matching scores from RANSAC on SSSG and, lastly, matching scores for 3D-NDT histograms. Since the methods examined here actually return distances, i.e., Euclidean distance between histograms in the case of 3D-NDT-histograms, and mean distance between candidate matching inliers for SSSG-RANSAC, the brightness values in the figure are inverted to be visually comparable to the overlap. On first inspection, it appears that we can observe some general similarities between the predictions and ground truth place correspondences for the SSSG-RANSAC method, but that the case is not as clear for the 3D-NDT histograms.

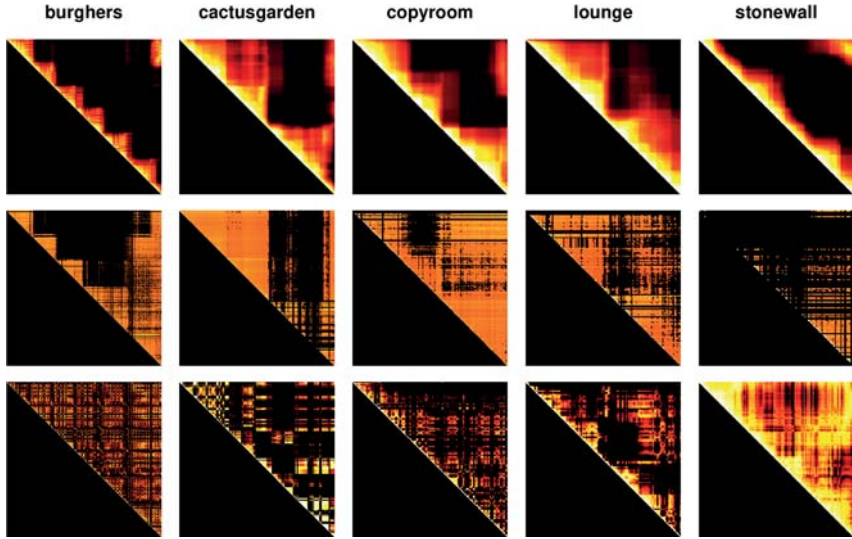


Figure 6.5: The top row shows the overlap ratio for each pose relative to all previous poses (brighter is higher and black indicates zero overlap). The second row shows the SSSG-RANSAC matching score (brighter indicates a lower distance between source and target models). Black indicates that none of the random samples resulted in a transformation that passed the inlier ratio test. The bottom row shows the respective results using 3D-NDT-histogram matching. Each column refers to a different scene from the Stanford scene data-set. Since the criterion that determines whether two places are the same location is based on overlap, a successful place-recognition system would appear similar to the top row.

Recalling the definitions of True Positive Rate (TPR) and False Positive Rate (FPR) as:

$$\text{TPR} = \frac{\sum \text{true positive}}{\sum \text{condition positive}} \quad (6.7)$$

and

$$\text{FPR} = \frac{\sum \text{false positive}}{\sum \text{condition negative}} \quad (6.8)$$

respectively, let us plot the receiver operating characteristics (ROC) of the two classifiers, shown in Fig. 6.6. This, at a minimum, shows a performance that appears better than random for both classifiers, with SSSG-RANSAC returning roughly twice as many true positive matches for every false positive, compared to

3D-NDT-histograms. For both tests we observe that an increase in the threshold for accepting a pair of scenes as matching causes the true positive rate to rise at a higher rate than the false positive rate. However for the SSSG-RANSAC approach there is a point at which the inlier ratio test begins to reject matches outright, and the curve ends abruptly. Suppose that a classifier has indicated a

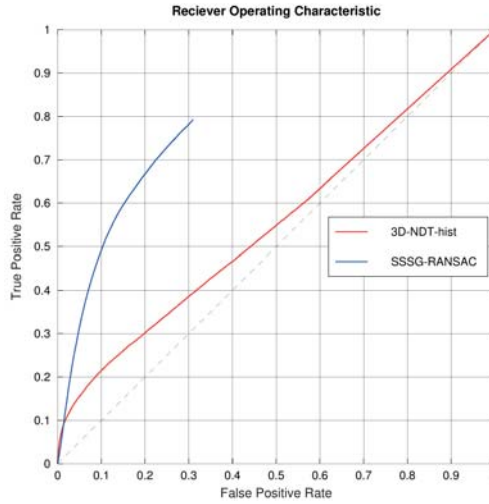


Figure 6.6: True positive rate (TPR) vs. false positive rate (FPR) plotted for varying thresholds of the classifiers. By lowering the threshold, more matches are reported. However, this comes at a cost of mislabeling a larger proportion of pairs (indicated by the increase in FPR). The diagonal line is the 50% ratio which represents the performance of a classifier that randomly assigns labels, in proportion to their true frequency e.g. guessing positive half the time, in a situation where matching and non-matching pairs are equally numerous.

match, how safe would it be to trust it? In applications such as SLAM it is often worse to include a false positive and distort a map that may otherwise not have had serious issues than to miss a large number of potential matches. Another way to phrase the question is to ask what is the precision (or positive predictive value) of the classifier? This quantity is defined as:

$$\text{precision} = \frac{\sum \text{true positive}}{\sum \text{prediction positive}} \quad (6.9)$$

In other words, out of all the times that the classifier made a positive prediction, how many times was it actually right? By looking at the precision curve in Fig. 6.7 it seems the answer is “often”. Even at its worst, the SSSG-RANSAC classifier is correct close to 38% of the times when indicating a match. To put

this figure into perspective, we ought to consider that out of all pairs, only 19.5% of the pairs are actually overlapping sufficiently to be considered the same location. This sounds good, but compared to the number of poses that

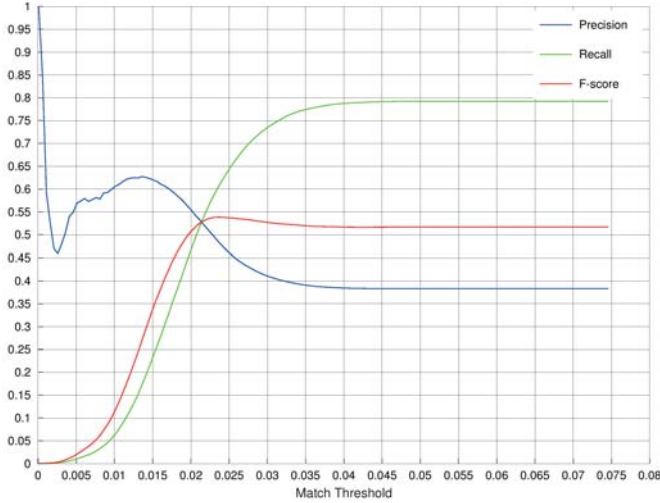


Figure 6.7: Precision, recall and F-score for the SSSG-RANSAC classifier. The graph is truncated on the ordinate axis since the curves stabilize at 0.05. Because the RANSAC algorithm sets a limit on the permitted ratio of outliers when searching for a solution, no additional matches tend to be reported beyond a certain match threshold.

actually correspond to the same location, how often is the classifier correctly identifying them? This is quantified by the *recall*, or probability of detection, defined as:

$$\text{recall} = \frac{\sum \text{true positive}}{\sum \text{condition positive}} \quad (6.10)$$

Recall starts at a very low ratio and increases at a slower pace than the precision deteriorates. When the match threshold is at 0.015 the classifier is only detecting a quarter of the total matches, but the precision has already decreased by over 30%. At the point where the precision and recall curves cross, SSSG-RANSAC is finding half the positive matches, but is wrong approximately half the times it reports a match. Additionally, we can consider a single score, i.e., the F-score (also known as the F_1 – score), computed as:

$$\text{F-score} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (6.11)$$

The highest F-score occurs at approximately 0.022 and represents a reasonable trade-off between precision and recall, in the absence of specific preferences in favor of one or the other.

In Fig. 6.8 we can see the precision, recall and F-score for 3D-NDT-histogram matching. There we see a precision curve that appears to promise a much higher precision for low recall than the SSSG-RANSAC method. If we care more about quality than quantity this method seems to be the best option among the two. For example, when finding 5% of the matches, it is outputting a correct classification over 70% of the time, compared to a mere 60% precision of SSSG-RANSAC. This analysis is somewhat misleading if our interest is chiefly in loop-detection,

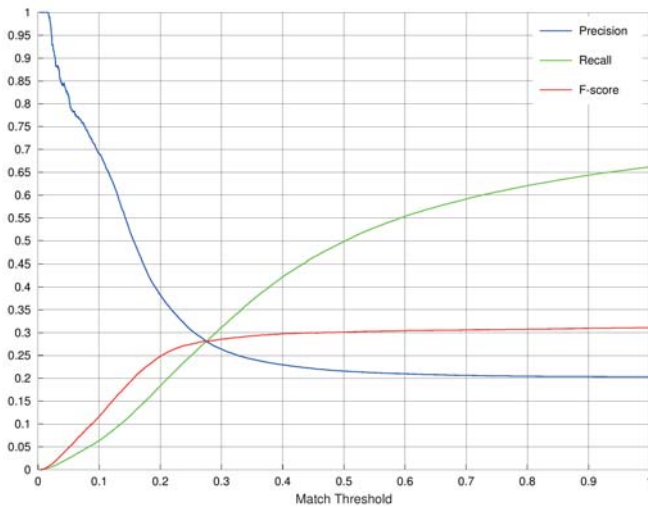


Figure 6.8: Precision, recall and F-score for 3D-NDT-histogram matching. We see the precision stabilize at roughly 20% while the recall increases. Since the percentage of pairs that are matching in the data-set is 19.5%, this corresponds to the fraction of times we should expect to be correct if simply assuming every pair was a match

however. If we disallow matching between nearby pairs of frames and compute the precision and recall values for the two classifiers, e.g. considering frames that are separated by an index of at least 20 (essentially removing the 20 diagonals closest to the main diagonal from the matrices in Fig. 6.5) we see the precision at 5% recall drop to 16% and 44% for the 3D-NDT-histograms and SSSG-RANSAC methods, respectively. Given that the remaining pairs with condition positive only represent 11.5% of the total we find that the 3D-NDT-histogram matching is barely outperforming guessing for non-adjacent pairs.

Pose Errors

The output of SSSG-RANSAC is a hypothesized rigid-body transformation in addition to the residual associated with it. We can compare these hypotheses to independently derived ground-truth pose estimates, obtained by jointly optimizing over all sensor poses in a global fashion, using all the available data in the RGB and depth images.

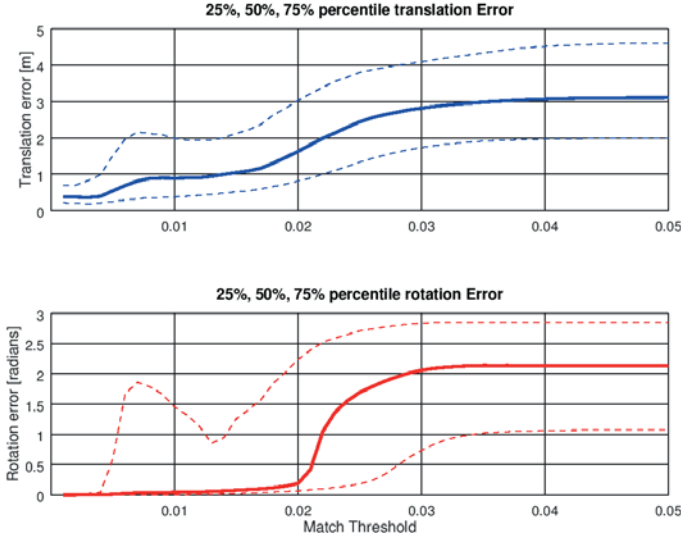


Figure 6.9: Translation and Rotation errors, relative to ground truth poses shown as different percentiles. There are no errors larger than 5m in translation, since the reconstructed TSDF volume is 5m on each side. The largest misalignment thus has to be strictly smaller than this. Angular error is bounded by π radians since any larger angle would bring the geometry back into alignment again.

In Fig. 6.9 we see the median pose errors for the true positive matches produced by the SSSG-RANSAC method. Translation is represented as the norm of the difference between best RANSAC hypothesis translation and the ground-truth. The rotation is computed by obtaining the angle-axis representation of the relative rotation:

$$R_{ab} = R_b^{-1} R_a \quad (6.12)$$

The figure also tells us that lower matching thresholds are linked to lower pose errors. Since the graph is sparse, a rough initial alignment may still be sufficient, for subsequent ICP-like algorithms to converge to the global optimum, since the closest point may be an actual corresponding vertex.

6.3 Discussion

We've presented a place-recognition system based on RANSAC, applied to sparse stable scene graphs. Place recognition based only on global geometric structure is challenging and without the abundant information present in texture one is given precious few clues to work with. Nonetheless, the experimental results show that the approach is feasible, with plenty of room for improvement. Different sensor systems, such as omni-directional LIDAR arrays, may observe larger portions of the environment at a time, leading to greater overlap between scans and consequently performance.

The TSDF allows for the integration of more data than any single depth image is likely to contain, thus compensating for the narrow field of view of typical camera-like sensors. Encoding the salient features and the relationship between them into a graph is a way of summarizing the essential geometric structure of the TSDF. By its construction, it evidently lacks information about the orientation and extent of surfaces, other than that which is implied by the loops that are occasionally made by the SSSG edges. Orientations may be an important characteristic to preserve for place recognition and one could possibly extend the concept of SSSG to reason explicitly about and promote the formation of triangular patches within the graph.

Although we stated earlier that the method used for generating the SSSG is independent of the process that generates the feature points, there are potential benefits to using image-based algorithms for obtaining them. Applying a method designed for images on the entire volume has the effect of producing a feature response value at every voxel. In the case of features that respond to curvature, for example, the feature response can be used to distinguish features that are joined by flat and curved geometry. In Fig. 6.10 we show an example application where the SSSG is extracted from the TSDF reconstruction of an office desk. The purple edges are those that pass through the surface and simultaneously have a high feature response along their extent. For performance, implementation

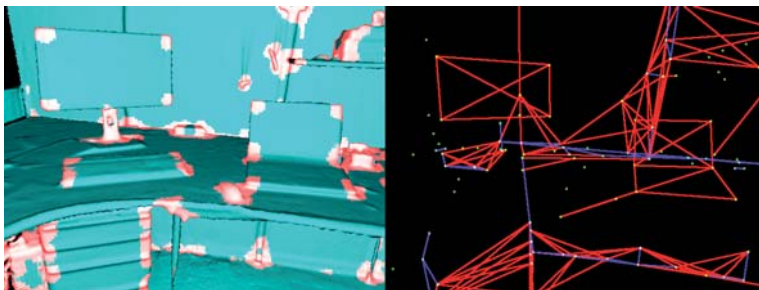


Figure 6.10: Reconstruction with feature response and SSSG wherein edges have been differentiated based on the mean feature response along their length.

and interfacing reasons the experiments in this chapter employed a point-cloud based Harris detector and therefore could not integrate this information in the SSSG-RANSAC matcher.

6.3.1 Improvements and Future work

There is potential for improvement on these methods that has not been fully explored due to temporal constraints. For instance, the number of iterations needed to reliably match graphs by RANSAC is somewhat limiting for real-time applications. It would be beneficial to have a pre-screening process to filter out unlikely matches, so that more resources can be spent on rejecting more likely pairs. The standard approach would be to limit the selected points for generating models to pair up with RANSAC, by trying to explicitly select correspondences based on feature descriptors that could be stored at each vertex. This approach imposes an ordering on the samples such that the most similar pairs of triplets are picked first [137]. Since the geometric nature of the problem is invariant to scale changes, one can also use the similarity between perimeter and area of the resulting triangles as further rejection criteria. Which descriptors would serve best in this scenario is still an open question, but we have many to choose from. At each graph vertex we can store any of the local depth image and volumetric feature descriptors, as discussed in Chapters 4 and 5. Since we can extract a surface interface from the TSDF one of many 3D point-cloud features [138] can be used, too. We are also free to compute descriptors on the graph itself, such as Heat Kernel Signatures [139], Wave Kernel Signatures [140], and other local graph descriptors. One may additionally adapt point-cloud descriptors to work on graphs, making potential savings in computational complexity by using the neighborhood relations implied by the edge connectivity instead of searching for nearest neighbors. An interesting candidate descriptor for application on graphs is the Histograms of Distances [141].

Modeling the observed space as a graph enables us to make use of general ⁵ graph matching algorithms, popular among which are spectral methods [142, 143, 144, 145], which have not been tested so far.

Improvements to the stability of the SSSG may also be achieved by obtaining better feature detectors. For example, inspired by the FAST-ER [146] method, a machine-learning approach to corner detection, one could possibly train a classifier to output a feature response based on a neighborhood of voxels. While this may not significantly outperform a corner detector such as Harris corners, it could be made to perform more selectively, e.g. responding only to corners where surfaces meet at right angles or even be made to factor out specific artifacts caused by projectively fusing depth images into the TSDF.

⁵A weighted graph can trivially be constructed from the SSSG, by inputting the length of edges into an adjacency matrix. To obtain a directed graph, some consistent convention would need to be followed, which may prove to be more challenging

Chapter 7

Conclusion

In this thesis we have taken, as a point of departure, the TSDF representation and the range image integration algorithm of Curless et al. [16, 20] and investigated its applicability to the field of robotics as a map representation. The TSDF accurately captures the mean and variance of the measurements used to generate it and was shown, by Newcombe et al. [21], that the TSDF provides an efficient way of generating high-quality depth images for frame-to-frame tracking. We proposed an alternative formulation of the tracking problem that uses point-to-model distances leading to an implementation better suited for use on CPU hardware. Building on this result, we investigated the potential benefits of TSDFs to aid in object and place recognition tasks and found TSDF maps to be of instrumental benefit to robots involved in shape-based recognition tasks. Having presented a case for the adoption of TSDF maps in robotics, we then focus on mitigating their most obvious shortcomings in memory and computational complexity. We show that through unsupervised learning algorithms based on PCA and artificial neural networks, constant-time lossy compression schemes can be devised to virtually extend mapped volumes from small rooms to much larger environments, with potentially beneficial side-effects in terms of noise-removal. Finally, we propose a derived map representation that can be applied to even larger-scales or, as explored in this thesis, to place recognition. We review these contributions in the sections 7.1, 7.2, and 7.3. In section 7.4, we discuss future work.

7.1 A Practical Guide to TSDF Mapping

While signed distance fields (or functions) are not in themselves an uncommon occurrence in mathematics and computer graphics their use as a map representation in robotics is a relatively recent development. In spite of its increasing popularity, the problem of reconstructing a discrete volumetric TSDF from a set of depth measurements involves several methods and assumptions that are not always clearly stated, understood nor questioned in the literature. Among

these we can count the truncation distance, its relationship with grid resolution and sensor variance, the number of bits per voxel, as well as interpolation and gradient estimation methods; these design choices are all investigated in this thesis. For the benefit of those who desire an introduction to the topic as well as those who require a reliable reference guide, one of the main contributions of this thesis is an accessible overview to TSDFs and an in-depth analysis of their different properties. To relate the various parameter choices to concrete implications, we investigate the impact they have on the task of pose-tracking and surface mapping.

7.2 Large-Scale Tracking and Mapping

While different approaches have been aimed at extending the working volume of TSDF models for mapping applications this thesis presents a first use of a fast single-pass compression algorithm for this purpose. We arrived at this approach by investigating unsupervised machine-learning methods using auto-encoder networks and PCA-based dictionaries. Although the resulting compression method is lossy, experiments indicate that much of the discarded content is high-frequency noise. Consequently, tracking depth-cameras against maps that have undergone compression often results in more stable performance than tracking against the original map. The ability to re-integrate previously mapped sections that have been compressed, back into the original TSDF representation allows this method to be used as a seamless modular replacement for the regular representation. Since the tracking algorithm proposed in Chapter 3 is designed to run on a multi-core CPU, the compression algorithm was instead designed to exploit the parallel functionality of GPUs to allow tracking, mapping and compression to run simultaneously on a single system in real-time scenarios. The proposed volumetric compression method is not useful only for our particular application, however. A number of interesting algorithms in robotics are already based on voxel representations and can be similarly extended, to enable either larger maps or higher resolution, whilst simultaneously dealing with noisy input.

7.3 TSDFs and Shape-based Features

Much of the work in this thesis has focused on the evaluation of feature detectors and descriptors due to their common use in object and place recognition pipelines. These evaluations are counted among our contributions insofar as they offer useful guidelines for researchers, students and engineers faced with making choices between many available options.

It is known that the underrepresentation of sharp corners in sample-based TSDFs is a weakness, as described in Chapter 2. However, we find that feature detectors and descriptors are, respectively, more stable and more reliably matched on depth images rendered from TSDF than on raw depth images from a physical sensor. Porting standard computer vision algorithms based on 2D image

pixels with intensity content to 3D image voxels containing signed distances is shown to be another viable strategy, as evidenced by our implementation and evaluation of Harris corners in TSDFs. However, we also show that integral invariant features, specifically designed for signed distance fields break down in truncated and discrete settings (and even in their intended application domain, for some overlooked cases). These evaluations provide insights for further research in shape-based perception.

This thesis also proposes to go beyond standard methods involving local geometric descriptors by describing the mapped geometry as a scene graph of interconnected salient features. Using a global alignment strategy for matching these graphs results in much better scores than related work on global shape descriptors based on 3D normal distributions transforms. The results of these experiments indicate that place recognition by means of global shape description is a difficult problem. Though the presented study does offer some directions for further improvement.

7.4 Future Work

Although much has been written in this thesis regarding the TSDF representation itself, there are several points of view on the subject that have not been explored. Curless originally derived the TSDF integration algorithm used here as a maximum likelihood estimate for the surface position [16]. Among the assumptions used in that derivation were: that each measurement ray is statistically independent, that the noise-model of the sensor is Gaussian, that the sensor alignment errors are much smaller than the measurement errors, and that these measurement errors are distributed along the line of sight of each ray. As sensor characteristics change, it may be worth to review these assumptions to find more fitting formulations for the volumetric integration process of TSDFs.

The recent popularity of TSDFs is arguably owed to the KinectFusion algorithm, proposed by Newcombe et al. [21] in 2011. And there have been many extensions proposed to it. In fact, one may view the tracking algorithm [53] from Chapter 3 as one among them. The modular way in which the topic of TSDFs for tracking and mapping applications has been explored in this thesis is useful, since it allows one to look specifically at the impact of bit-rates, interpolation strategies, gradient estimation, and compression. However, we have yet to evaluate the effect of several of these (and other!) proposed choices in combination.

The challenge of shape-based place recognition remains an open problem in confined spaces with limited field of view. However machine learning methods that use TSDF volumes as input achieve object detection with remarkable robustness to occlusions and measurement noise [147]. Other object recognition systems have been proposed, using volumetric occupancy [148, 149]. These methods are applicable to TSDF volumes as well by converting to an approximate occupancy, as described in Chapter 2. While these methods, including our

compression algorithms from Chapter 5 all work on small chunks of volumetric data, they could be adapted to yield novel whole-scene descriptors for place recognition. An interesting future work is to employ such scene descriptors to filter candidate locations for place recognition, reducing the number of subsequent steps using e.g. RANSAC.

7.5 Closing Remarks

The TSDF is a powerful and versatile representation of 3D space, that lends itself to many different applications and extensions. However, the impression that remains with me, at the end of this thesis is that we are not likely to find one map representation *to rule them all*, so to speak. Even if one can find a representation that on average performs well on a wide range of tasks, there is likely to be a better solution from heterogeneous representations that may not require compromises between competing objectives. This may entail using highly specialized views of the world that are suitable for particular problems. For example, using sparse landmarks for localization, dense mapping for precise manipulation or 2D color images for object recognition. An amalgam of different mapping paradigms may thus be necessary for robots to reach human-level performance on complex tasks that involve several different subtasks from a variety of problem domains.

On the other hand, as we explore the possibilities that open up given dense volumetric modeling, we may even discover entirely new problem domains that become addressable. One might for example envision connecting dense 3D mapping to multi-physics simulation, allowing robots to make accurate predictions in the field that would be infeasible given even the best of human intuition and reasoning.

The answer to whether a TSDF *can be used to improve the mapping, perception and recognition capabilities* of mobile autonomous robots is, as for any question worth investigating, inconclusive. In a warehouse designed for robots, with tags and markers spaced out at regular intervals and bar-codes on every relevant object, the answer is trivially negative. In an environment where precise maneuvering or manipulation needs to be performed, but recognizable markers are absent, the TSDF may provide certain benefits over (and in combination with) other methods. If visual sensing is barred by environmental factors or perhaps by privacy concerns, it may be among the top candidate choices within the current state of the art.

Appendices

Appendix A

Code listings

A.1 C++ code for packing multiple variables into a single byte

This code listing provides an example implementation in C++ of packing two values into a single byte using a bitfield. Alternatively, it also shows how one can store the two values as a single variable e.g. an unsigned byte, and use bit shifts and bit-masks for reading and writing

```
// Arrays of bitfields and bytes
#include <iostream>
#include <string>
#include <ctime>
#include <cmath>

#define numBits 4

constexpr unsigned int maxVal = pow(2,numBits)—1;

#pragma pack(push,1)
struct voxel{
    unsigned D : numBits;
    unsigned W : numBits;
};
#pragma pack(pop)

unsigned int bitsFromFloat(const float input){
    return std::min<unsigned int>(maxVal, std::rint(maxVal*input));
}

float floatFromBits(const unsigned int input){
    return input/Float(maxVal);
}

unsigned int readD(const unsigned char byteVoxel){
    return byteVoxel >> 4;
}

unsigned int readW(const unsigned char byteVoxel){
    return byteVoxel & 0b00001111;
}

void writeD(unsigned char &byteVoxel, const unsigned char input){
    byteVoxel = byteVoxel & 0b00001111; //clear 4 MSB
    byteVoxel = byteVoxel | (input << 4); //write
}

void writeW(unsigned char &byteVoxel, const unsigned char input){
    byteVoxel = byteVoxel & 0b11110000; //clear 4 LSB
    byteVoxel = byteVoxel | (input & 0b00001111); //write
}
```

```

int main()
{
    constexpr auto N = 10u;
    voxel bitfieldArray[N];
    unsigned char byteArray[N];

    std::srand(std::time(0));
    for(auto idx = 0u; idx < N; ++idx)
    {
        float d = float(std::rand())/RAND_MAX;
        float w = float(std::rand())/RAND_MAX;

        std::cout << "inserting_"
        << d << "_and_"
        << w << "_as_"
        << bitsFromFloat(d) << "_and_"
        << bitsFromFloat(w) << "\n";

        bitfieldArray[idx].D = bitsFromFloat(d);
        bitfieldArray[idx].W = bitsFromFloat(w);

        writeD(byteArray[idx], bitsFromFloat(d));
        writeW(byteArray[idx], bitsFromFloat(w));
    }

    std::cout << std::endl;
    std::cout << "bitfieldArray_requires_" << sizeof(bitfieldArray) << "_bytes\n";
    std::cout << "byteArray_requires_" << sizeof(byteArray) << "_bytes\n\n";

    for(auto idx = 0u; idx < N; ++idx)
    {
        std::cout << idx << "_bitfield_ as int:_"
        << readD(byteArray[idx]) << ",_"
        << readW(byteArray[idx])
        << "\t as float:_"
        << floatFromBits(readD(byteArray[idx])) << ",_"
        << floatFromBits(readW(byteArray[idx])) << "\n";

        std::cout << idx << "_byte_ as int:_"
        << bitfieldArray[idx].D << ",_"
        << bitfieldArray[idx].W
        << "\t as float:_"
        << floatFromBits(bitfieldArray[idx].D) << ",_"
        << floatFromBits(bitfieldArray[idx].W) << "\n";
    }
}

```

Example output:

```

inserting 0.631785 and 0.617026 as 9 and 9
inserting 0.852161 and 0.147901 as 13 and 2
inserting 0.0856126 and 0.415967 as 1 and 6
inserting 0.78367 and 0.153177 as 12 and 2
inserting 0.212242 and 0.687655 as 3 and 10
inserting 0.572266 and 0.49504 as 9 and 7
inserting 0.892832 and 0.799307 as 13 and 12
inserting 0.465086 and 0.862731 as 7 and 13
inserting 0.145797 and 0.230016 as 2 and 3
inserting 0.486525 and 0.215539 as 7 and 3

bitfieldArray requires 10 bytes
byteArray requires 10 bytes

0 bitfield -- as int: 9, 9 as float: 0.6, 0.6
0 byte -- as int: 9, 9 as float: 0.6, 0.6
1 bitfield -- as int: 13, 2 as float: 0.866667, 0.133333
1 byte -- as int: 13, 2 as float: 0.866667, 0.133333
2 bitfield -- as int: 1, 6 as float: 0.0666667, 0.4
2 byte -- as int: 1, 6 as float: 0.0666667, 0.4
3 bitfield -- as int: 12, 2 as float: 0.8, 0.133333
3 byte -- as int: 12, 2 as float: 0.8, 0.133333
4 bitfield -- as int: 3, 10 as float: 0.2, 0.666667
4 byte -- as int: 3, 10 as float: 0.2, 0.666667
5 bitfield -- as int: 9, 7 as float: 0.6, 0.466667
5 byte -- as int: 9, 7 as float: 0.6, 0.466667
6 bitfield -- as int: 13, 12 as float: 0.866667, 0.8
6 byte -- as int: 13, 12 as float: 0.866667, 0.8
7 bitfield -- as int: 7, 13 as float: 0.466667, 0.866667
7 byte -- as int: 7, 13 as float: 0.466667, 0.866667
8 bitfield -- as int: 2, 3 as float: 0.133333, 0.2
8 byte -- as int: 2, 3 as float: 0.133333, 0.2
9 bitfield -- as int: 7, 3 as float: 0.466667, 0.2
9 byte -- as int: 7, 3 as float: 0.466667, 0.2

```


References

- [1] E. AB, “Electrolux unveils prototype for robot vacuum cleaner (press-release),” Dec 1997.
- [2] V. D. Ek, “Robotgrasklipparen pa stark framatmarsch,” Jul 2016.
- [3] E. S. Larsen and D. McAllister, “Fast matrix multiplies using graphics hardware,” in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pp. 55–55, ACM, 2001.
- [4] R. Susskind and D. Susskind, *The future of the professions: How technology will transform the work of human experts*. Oxford University Press, USA, 2015.
- [5] P. Van Parijs, “Basic income: a simple and powerful idea for the twenty-first century,” *Politics & Society*, vol. 32, no. 1, pp. 7–39, 2004.
- [6] R. Sparrow, “Killer robots,” *Journal of applied philosophy*, vol. 24, no. 1, pp. 62–77, 2007.
- [7] F. Sauer and N. Schörnig, “Killer drones: The ‘silver bullet’ of democratic warfare?,” *Security Dialogue*, vol. 43, no. 4, pp. 363–380, 2012.
- [8] R. C. Arkin, “Ethical robots in warfare,” *IEEE Technology and Society Magazine*, vol. 28, no. 1, pp. 30–33, 2009.
- [9] N. Bostrom, “In defense of posthuman dignity,” *Bioethics*, vol. 19, no. 3, pp. 202–214, 2005.
- [10] Council of European Union, “Council regulation (EU) no 428/2009,” 2009.
<http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32009R0428>.
- [11] J. P. Morgan and R. L. Tutwiler, “Real-time reconstruction of depth sequences using signed distance functions,” in *SPIE Defense+ Security*,

- pp. 909117–909117, International Society for Optics and Photonics, 2014.
- [12] J. A. Ratches, “Review of current aided/automatic target acquisition technology for military target acquisition tasks,” *Optical Engineering*, vol. 50, no. 7, pp. 072001–072001, 2011.
 - [13] A. Harvey, “Cv dazzle: Camouflage from computer vision,” *Technical report*, 2012.
 - [14] A. Elfes, “Occupancy grids: A probabilistic framework for robot perception and navigation,” 1989.
 - [15] C. Hernández, G. Vogiatzis, and R. Cipolla, “Probabilistic visibility for multi-view stereo,” in *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*, pp. 1–8, IEEE, 2007.
 - [16] B. L. Curless, *New methods for surface reconstruction from range images*. PhD thesis, Stanford University, 1997.
 - [17] P. Bloomfield and W. Steiger, *Least absolute deviations: Theory, applications and algorithms*, vol. 6. Springer Science & Business Media, 2012.
 - [18] P. Mullen, F. De Goes, M. Desbrun, D. Cohen-Steiner, and P. Alliez, “Signing the unsigned: Robust surface reconstruction from raw pointsets,” in *Computer Graphics Forum*, vol. 29, pp. 1733–1741, Wiley Online Library, 2010.
 - [19] H. Xu and J. Barbič, “Signed distance fields for polygon soup meshes,” in *Proceedings of Graphics Interface 2014*, pp. 35–41, Canadian Information Processing Society, 2014.
 - [20] B. Curless and M. Levoy, “A volumetric method for building complex models from range images,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, ACM, 1996.
 - [21] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, “KinectFusion: Real-time dense surface mapping and tracking,” in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pp. 127–136, 2011.
 - [22] M. D. Wheeler, Y. Sato, and K. Ikeuchi, “Consensus surfaces for modeling 3d objects from multiple range images,” in *Computer Vision, 1998. Sixth International Conference on*, pp. 917–924, IEEE, 1998.

- [23] S. F. Frisken and R. N. Perry, "Efficient estimation of 3d euclidean distance fields from 2d range images," in *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pp. 81–88, IEEE Press, 2002.
- [24] J. C. Hart, "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.
- [25] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, 1975.
- [26] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *ACM siggraph computer graphics*, vol. 21, pp. 163–169, ACM, 1987.
- [27] G. M. Treece, R. W. Prager, and A. H. Gee, "Regularised marching tetrahedra: improved iso-surface extraction," *Computers & Graphics*, vol. 23, no. 4, pp. 583–598, 1999.
- [28] P. Stein, "A note on the volume of a simplex," *The American Mathematical Monthly*, vol. 73, no. 3, pp. 299–301, 1966.
- [29] K. Kanamori, T. Fumoto, and H. Kotera, "A color transformation algorithm using prism interpolation," in *IS&T's 8th International Congress on Advances in NIP Technologies*, pp. 164–174, 1992.
- [30] R. Barnhill, G. Birkhoff, and W. J. Gordon, "Smooth interpolation in triangles," *Journal of Approximation Theory*, vol. 8, no. 2, pp. 114–128, 1973.
- [31] G. M. Nielson, "The side-vertex method for interpolation in triangles," *Journal of Approximation theory*, vol. 25, no. 4, pp. 318–336, 1979.
- [32] P. E. Franklin, "Interpolation methods and apparatus," June 8 1982. US Patent 4,334,240.
- [33] K. Kanamori, H. Kawakami, and H. Kotera, "Novel color transformation algorithm and its applications," in *Electronic Imaging'90, Santa Clara, 11-16 Feb'92*, pp. 272–281, International Society for Optics and Photonics, 1990.
- [34] P. Hemingway, "n-simplex interpolation," *HP Laboratories Bristol, HPL-2002-320*, pp. 1–8, 2002.
- [35] E. Baumann, "Splitting a cube in tetrahedras," 2004. <http://www.baumanneduard.ch/Splittingacubeintetrahedras2.htm>, accessed 2017-03-09.

- [36] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, “Elasticfusion: Real-time dense slam and light source estimation,” *The International Journal of Robotics Research*, p. 0278364916669237, 2016.
- [37] K. J. Lee, Q. Zhao, X. Tong, M. Gong, S. Izadi, S. U. Lee, P. Tan, and S. Lin, “Estimation of intrinsic image sequences from image+ depth video,” in *European Conference on Computer Vision*, pp. 327–340, Springer, 2012.
- [38] R. Landaverde, T. Zhang, A. K. Coskun, and M. Herbordt, “An investigation of unified memory access performance in cuda,” in *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, pp. 1–6, IEEE, 2014.
- [39] R. A. Newcombe, *Dense Visual SLAM*. PhD thesis, Imperial College London, United Kingdom, June 2014.
- [40] C. Batty, “Problem: Reconstructing meshes with sharp features from signed distance data.” https://cs.uwaterloo.ca/~c2batty/misc/levelset_meshing/level_set_reconstruction.html, accessed 2017-01-10.
- [41] T. Ju, F. Losasso, S. Schaefer, and J. Warren, “Dual contouring of hermite data,” in *ACM Transactions on Graphics (TOG)*, vol. 21, pp. 339–346, ACM, 2002.
- [42] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, “Feature sensitive surface extraction from volume data,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 57–66, ACM, 2001.
- [43] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, “Real-time camera tracking and 3d reconstruction using signed distance functions,” in *Proceedings of Robotics: Science and Systems*, (Berlin, Germany), June 2013.
- [44] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, “Real-time 3d reconstruction in dynamic scenes using point-based fusion,” in *3DTV-Conference, 2013 International Conference on*, pp. 1–8, IEEE, 2013.
- [45] D. Lefloch, T. Weyrich, and A. Kolb, “Anisotropic point-based fusion,” in *Information Fusion (Fusion), 2015 18th International Conference on*, pp. 2121–2128, IEEE, 2015.

- [46] M. Meilland and A. I. Comport, “Super-resolution 3d tracking and mapping,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 5717–5723, IEEE, 2013.
- [47] J. Zienkiewicz, A. Tsiotsios, A. Davison, and S. Leutenegger, “Monocular, real-time surface reconstruction using dynamic level of detail,” in *3D Vision (3DV), 2016 Fourth International Conference on*, pp. 37–46, IEEE, 2016.
- [48] O. J. Woodman, “An introduction to inertial navigation,” tech. rep., University of Cambridge, Computer Laboratory, 2007.
- [49] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Robotics-DL tentative*, pp. 586–606, International Society for Optics and Photonics, 1992.
- [50] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pp. 145–152, IEEE, 2001.
- [51] B. D. Lucas, T. Kanade, *et al.*, “An iterative image registration technique with an application to stereo vision.,” in *IJCAI*, vol. 81, pp. 674–679, 1981.
- [52] M. Slavcheva, W. Kehl, N. Navab, and S. Ilic, “Sdf-2-sdf: Highly accurate 3d object reconstruction,” in *European Conference on Computer Vision*, pp. 680–696, Springer, 2016.
- [53] D. Ricao Canelhas, “Scene representation, registration and object detection in a truncated signed distance function representation of 3d space,” Master’s thesis, Örebro University, 2012.
- [54] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal, “Sdf tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3671–3676, IEEE, 2013.
- [55] A. W. Fitzgibbon, “Robust Registration of 2D and 3D Point Sets,” *Image and Vision Computing*, vol. 21, no. 13, pp. 1145–1153, 2003.
- [56] S. Baker and I. Matthews, “Lucas-kanade 20 years on: A unifying framework,” *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.
- [57] Z. Zhang, “Parameter estimation techniques: A tutorial with application to conic fitting,” *Image and vision Computing*, vol. 15, no. 1, pp. 59–76, 1997.

- [58] A. N. Tikhonov, "On the stability of inverse problems," in *Dokl. Akad. Nauk SSSR*, vol. 39, pp. 195–198, 1943.
- [59] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the IEEE Int. Conf. on Intelligent Robot Systems (IROS)*, 2012.
- [60] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, IEEE, 2011.
- [61] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3d visual slam with a hand-held rgb-d camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at Euron Forum*, 2011.
- [62] H. Andreasson and T. Stoyanov, "Real time registration of rgb-d data using local visual features and 3d-ndt registration," in *SPME Workshop at Int. Conf. on Robotics and Automation (ICRA)*, 2012.
- [63] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.
- [64] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?," in *European Conference on Computer Vision*, pp. 222–235, Springer, 2012.
- [65] P. Stotko, "State of the art in real-time registration of rgb-d images," 2016.
- [66] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," *ACM SIGGRAPH computer graphics*, vol. 20, no. 4, pp. 151–160, 1986.
- [67] M. Rouhani and A. D. Sappa, "Non-rigid shape registration: A single linear least squares framework," in *European Conference on Computer Vision*, pp. 264–277, Springer, 2012.
- [68] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 343–352, 2015.
- [69] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger, "Volumedeform: Real-time volumetric non-rigid reconstruction," in *European Conference on Computer Vision*, pp. 362–379, Springer, 2016.

- [70] J. Serafin and G. Grisetti, "Nicp: Dense normal based point cloud registration," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 742–749, IEEE, 2015.
- [71] R. B. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D registration," in *IEEE Int. Conf. on Robotics and Automation, ICRA 2009*, pp. 3212–3217, 2009.
- [72] B. Steder, G. Grisetti, and W. Burgard, "Robust place recognition for 3d range data based on point features," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 1400–1405, IEEE, 2010.
- [73] B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard, "Place Recognition in 3D Scans Using a Combination of Bag of Words and Point Feature based Relative Pose Estimation," in *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2011.
- [74] M. Magnusson, H. Andreasson, A. Nijchter, and A. J. Lilienthal, "Automatic appearance-based loop detection from 3D laser data using the normal distributions transform," *Journal of Field Robotics*, vol. 26, pp. 892–914, Nov. 2009.
- [75] L. Bo, X. Ren, and D. Fox, "Depth Kernel Descriptors for Object Recognition," in *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, pp. 821–826, IEEE, 2011.
- [76] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3D Recognition and Pose using the Viewpoint Feature Histogram," in *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, pp. 2155–2162, 2010.
- [77] H. Barrow and J. Tenenbaum, "Recovering intrinsic scene characteristics from images," *Computer Vision Systems*, pp. 3–26, 1978.
- [78] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images," in *Computer Vision, 1998. Sixth International Conference on*, pp. 839–846, IEEE, 1998.
- [79] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *Journal of Mathematical Imaging and Vision*, vol. 40, no. 1, pp. 120–145, 2011.
- [80] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "NARF: 3D Range Image Features for Object Recognition," in *Workshop on Realistic Perception in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.
- [81] S. Paris, P. Kornprobst, and J. Tumblin, *Bilateral filtering: Theory and applications*, vol. 1. Now Publishers Inc, 2009.

- [82] O. Woodford and G. Vogiatzis, "A generative model for online depth fusion," *Computer Vision–ECCV 2012*, pp. 144–157, 2012.
- [83] R. Hänsch, T. Weber, and O. Hellwich, "Comparison of 3d interest point detectors and descriptors for point cloud fusion," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 3, p. 57, 2014.
- [84] L. Bo, X. Ren, and D. Fox, "Kernel descriptors for visual recognition," in *Advances in neural information processing systems*, pp. 244–252, 2010.
- [85] P. Moreels and P. Perona, "Evaluation of Features Detectors and Descriptors based on 3D Objects," *International Journal of Computer Vision*, vol. 73, no. 3, pp. 263–284, 2007.
- [86] I. Gijbels, A. Lambert, and P. Qiu, "Edge-preserving image denoising and estimation of discontinuous surfaces," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 7, pp. 1075–1087, 2006.
- [87] S. G. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on image processing*, vol. 9, no. 9, pp. 1532–1546, 2000.
- [88] J.-L. Starck, E. J. Candès, and D. L. Donoho, "The curvelet transform for image denoising," *IEEE Transactions on image processing*, vol. 11, no. 6, pp. 670–684, 2002.
- [89] P. Jain and V. Tyagi, "A survey of edge-preserving image denoising methods," *Information Systems Frontiers*, vol. 18, no. 1, pp. 159–170, 2016.
- [90] N. J. Mitra and M. Pauly, "Shadow art," in *ACM Transactions on Graphics*, vol. 28, pp. 156–1, 2009.
- [91] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover," tech. rep., DTIC Document, 1980.
- [92] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, p. 50, Citeseer, 1988.
- [93] G. Willems, T. Tuytelaars, and L. Van Gool, "An efficient dense and scale-invariant spatio-temporal interest point detector," in *Computer Vision–ECCV 2008*, pp. 650–663, Springer, 2008.
- [94] I. Sobel and G. Feldman, "A 3x3 isotropic gradient operator for image processing," 1968.
- [95] E. Dam and B. ter Haar Romeny, "Front end vision and multi-scale image analysis," *Deep Structure I, II & III*, no. 1-4020, pp. 1507–0, 2003.

- [96] H. Scharr, *Optimal operators in digital image processing*. PhD thesis, 2000.
- [97] S. Manay, B.-W. Hong, A. J. Yezzi, and S. Soatto, *Integral invariant signatures*. Springer, 2004.
- [98] H. Pottmann, J. Wallner, Q.-X. Huang, and Y.-L. Yang, “Integral invariants for robust geometry processing,” *Computer Aided Geometric Design*, vol. 26, no. 1, pp. 37–60, 2009.
- [99] Q.-Y. Zhou and V. Koltun, “Dense scene reconstruction with points of interest,” *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 112, 2013.
- [100] I. E. Richardson, *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004.
- [101] M. W. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards, and Practice*, vol. 1. springer, 2002.
- [102] M. W. Jones, “Distance field compression,” 2004.
- [103] D. Meagher, “Geometric modeling using octree encoding,” *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.
- [104] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, “Adaptively sampled distance fields: a general representation of shape for computer graphics,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 249–254, ACM Press/Addison-Wesley Publishing Co., 2000.
- [105] S. Laine and T. Karras, “Efficient sparse voxel octrees,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 8, pp. 1048–1059, 2011.
- [106] M. Zeng, F. Zhao, J. Zheng, and X. Liu, “A memory-efficient kinectfusion using octree,” in *Computational Visual Media*, pp. 234–241, Springer, 2012.
- [107] F. Steinbrucker, C. Kerl, and D. Cremers, “Large-scale multi-resolution surface reconstruction from rgb-d sequences,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3264–3271, 2013.
- [108] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, “Real-time 3d reconstruction at scale using voxel hashing,” *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 169, 2013.

- [109] T. Stoyanov, R. Krug, R. Muthusamy, and V. Kyrki, “Grasp envelopes : Extracting constraints on gripper postures from online reconstructed 3d models,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* :, pp. 885–892, 2016.
- [110] H. Roth and M. Vona, “Moving volume kinectfusion.,” in *BMVC*, pp. 1–11, 2012.
- [111] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, “Kintinuous: Spatially extended KinectFusion,” in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, (Sydney, Australia), Jul 2012.
- [112] J. A. Sethian, “A fast marching level set method for monotonically advancing fronts,” *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.
- [113] H. Zhao, “A fast sweeping method for eikonal equations,” *Mathematics of computation*, vol. 74, no. 250, pp. 603–627, 2005.
- [114] M. Ruhnke, L. Bo, D. Fox, and W. Burgard, “Compact rgb-d surface models based on sparse coding.,” in *AAAI*, 2013.
- [115] M. Aharon, M. Elad, and A. Bruckstein, “-svd: An algorithm for designing overcomplete dictionaries for sparse representation,” *Signal Processing, IEEE Transactions on*, vol. 54, no. 11, pp. 4311–4322, 2006.
- [116] F. Bazyari and Y. Tzimiropoulos, “An active patch model for real world appearance reconstruction,” in *Workshop at the European Conference on Computer Vision*, pp. 443–456, Springer, 2014.
- [117] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1, pp. 37–52, 1987.
- [118] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Cognitive modeling*, 1988.
- [119] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [120] I. Quilez, “Modeling with distance functions,” 2008. online at <http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm>(accessed on Aug 2017).

- [121] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [122] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio, “Pylearn2: a machine learning research library,” *arXiv preprint arXiv:1308.4214*, 2013.
- [123] nVidia, “Cublas library,” 2008.
<https://developer.nvidia.com/cuBLAS> (accessed on Jun 2017).
- [124] J. Hoberock and N. Bell, “Thrust: A parallel template library,” 2010.
Online at <https://thrust.github.io> (accessed on Aug 2017).
- [125] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, “Deep convolutional inverse graphics network,” in *Advances in Neural Information Processing Systems*, pp. 2539–2547, 2015.
- [126] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal, “From feature detection in truncated signed distance fields to sparse stable scene graphs,” *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1148–1155, 2016.
- [127] R. T. Chin and C. R. Dyer, “Model-based recognition in robot vision,” *ACM Computing Surveys (CSUR)*, vol. 18, no. 1, pp. 67–108, 1986.
- [128] R. Nevatia and T. O. Binford, “Description and recognition of curved objects,” *Artificial Intelligence*, vol. 8, no. 1, pp. 77–98, 1977.
- [129] N. J. A. Sloane and Others, “The on-line encyclopedia of integer sequences (oeis)..” published electronically at <https://oeis.org>, Sequence A000217, 2009.
- [130] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [131] M. Magnusson, A. Lilienthal, and T. Duckett, “Scan registration for autonomous mining vehicles using 3d-ndt,” *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827, 2007.
- [132] T. Stoyanov, *Reliable Autonomous Navigation in Semi-Structured Environments using the Three-Dimensional Normal Distributions Transform (3D-NDT)*. PhD thesis, Örebro universitet, 2012.

- [133] M. Magnusson, *The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection*. PhD thesis, Örebro universitet, 2009.
- [134] J. Hannay and J. Nye, “Fibonacci numerical integration on a sphere,” *Journal of Physics A: Mathematical and General*, vol. 37, no. 48, p. 11591, 2004.
- [135] B. Keinert, M. Innmann, M. Sängner, and M. Stamminger, “Spherical fibonacci mapping,” *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 193, 2015.
- [136] B. K. Horn, “Closed-form solution of absolute orientation using unit quaternions,” *JOSA A*, vol. 4, no. 4, pp. 629–642, 1987.
- [137] O. Chum and J. Matas, “Matching with prosac-progressive sample consensus,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 220–226, IEEE, 2005.
- [138] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, and N. M. Kwok, “A comprehensive performance evaluation of 3d local feature descriptors,” *International Journal of Computer Vision*, vol. 116, no. 1, pp. 66–89, 2016.
- [139] J. Sun, M. Ovsjanikov, and L. Guibas, “A concise and provably informative multi-scale signature based on heat diffusion,” in *Computer graphics forum*, vol. 28, pp. 1383–1392, Wiley Online Library, 2009.
- [140] M. Aubry, U. Schlickewei, and D. Cremers, “The wave kernel signature: A quantum mechanical approach to shape analysis,” in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp. 1626–1633, IEEE, 2011.
- [141] O. Kechagias-Stamatis and N. Aouf, “Histogram of distances for local surface description,” in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 2487–2493, IEEE, 2016.
- [142] S. Umeyama, “An eigendecomposition approach to weighted graph matching problems,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 10, no. 5, pp. 695–703, 1988.
- [143] M. Gori, M. Maggini, and L. Sarti, “Exact and approximate graph matching using random walks,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 7, pp. 1100–1111, 2005.
- [144] T. Cour, P. Srinivasan, and J. Shi, “Balanced graph matching,” in *NIPS*, vol. 2, p. 6, 2006.

- [145] A. Egozi, Y. Keller, and H. Guterman, “A probabilistic approach to spectral graph matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 18–27, 2013.
- [146] E. Rosten, R. Porter, and T. Drummond, “Faster and better: A machine learning approach to corner detection,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 1, pp. 105–119, 2010.
- [147] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3d shapenets: A deep representation for volumetric shapes,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1912–1920, 2015.
- [148] S. Song and J. Xiao, “Deep sliding shapes for amodal 3d object detection in rgb-d images,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 808–816, 2016.
- [149] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 922–928, IEEE, 2015.

PUBLICATIONS *in the series*
ÖREBRO STUDIES IN TECHNOLOGY

1. Bergsten, Pontus (2001) *Observers and Controllers for Takagi – Sugeno Fuzzy Systems*. Doctoral Dissertation.
2. Iliev, Boyko (2002) *Minimum-time Sliding Mode Control of Robot Manipulators*. Licentiate Thesis.
3. Spännar, Jan (2002) *Grey box modelling for temperature estimation*. Licentiate Thesis.
4. Persson, Martin (2002) *A simulation environment for visual servoing*. Licentiate Thesis.
5. Boustedt, Katarina (2002) *Flip Chip for High Volume and Low Cost – Materials and Production Technology*. Licentiate Thesis.
6. Biel, Lena (2002) *Modeling of Perceptual Systems – A Sensor Fusion Model with Active Perception*. Licentiate Thesis.
7. Otterskog, Magnus (2002) *Produktionstest av mobiltelefonantennerna i mod-växlande kammare*. Licentiate Thesis.
8. Tolt, Gustav (2003) *Fuzzy-Similarity-Based Low-level Image Processing*. Licentiate Thesis.
9. Loutfi, Amy (2003) *Communicating Perceptions: Grounding Symbols to Artificial Olfactory Signals*. Licentiate Thesis.
10. Iliev, Boyko (2004) *Minimum-time Sliding Mode Control of Robot Manipulators*. Doctoral Dissertation.
11. Pettersson, Ola (2004) *Model-Free Execution Monitoring in Behavior-Based Mobile Robotics*. Doctoral Dissertation.
12. Överstam, Henrik (2004) *The Interdependence of Plastic Behaviour and Final Properties of Steel Wire, Analysed by the Finite Element Method*. Doctoral Dissertation.
13. Jennergren, Lars (2004) *Flexible Assembly of Ready-to-eat Meals*. Licentiate Thesis.
14. Jun, Li (2004) *Towards Online Learning of Reactive Behaviors in Mobile Robotics*. Licentiate Thesis.
15. Lindquist, Malin (2004) *Electronic Tongue for Water Quality Assessment*. Licentiate Thesis.
16. Wasik, Zbigniew (2005) *A Behavior-Based Control System for Mobile Manipulation*. Doctoral Dissertation.

17. Berntsson, Tomas (2005) *Replacement of Lead Baths with Environment Friendly Alternative Heat Treatment Processes in Steel Wire Production*. Licentiate Thesis.
18. Tolt, Gustav (2005) *Fuzzy Similarity-based Image Processing*. Doctoral Dissertation.
19. Munkevik, Per (2005) "Artificial sensory evaluation – appearance-based analysis of ready meals". Licentiate Thesis.
20. Buschka, Pär (2005) *An Investigation of Hybrid Maps for Mobile Robots*. Doctoral Dissertation.
21. Loutfi, Amy (2006) *Odour Recognition using Electronic Noses in Robotic and Intelligent Systems*. Doctoral Dissertation.
22. Gillström, Peter (2006) *Alternatives to Pickling; Preparation of Carbon and Low Alloyed Steel Wire Rod*. Doctoral Dissertation.
23. Li, Jun (2006) *Learning Reactive Behaviors with Constructive Neural Networks in Mobile Robotics*. Doctoral Dissertation.
24. Otterskog, Magnus (2006) *Propagation Environment Modeling Using Scattered Field Chamber*. Doctoral Dissertation.
25. Lindquist, Malin (2007) *Electronic Tongue for Water Quality Assessment*. Doctoral Dissertation.
26. Cielniak, Grzegorz (2007) *People Tracking by Mobile Robots using Thermal and Colour Vision*. Doctoral Dissertation.
27. Boustedt, Katarina (2007) *Flip Chip for High Frequency Applications – Materials Aspects*. Doctoral Dissertation.
28. Soron, Mikael (2007) *Robot System for Flexible 3D Friction Stir Welding*. Doctoral Dissertation.
29. Larsson, Sören (2008) *An industrial robot as carrier of a laser profile scanner. – Motion control, data capturing and path planning*. Doctoral Dissertation.
30. Persson, Martin (2008) *Semantic Mapping Using Virtual Sensors and Fusion of Aerial Images with Sensor Data from a Ground Vehicle*. Doctoral Dissertation.
31. Andreasson, Henrik (2008) *Local Visual Feature based Localisation and Mapping by Mobile Robots*. Doctoral Dissertation.
32. Bouguerra, Abdelbaki (2008) *Robust Execution of Robot Task-Plans: A Knowledge-based Approach*. Doctoral Dissertation.

33. Lundh, Robert (2009) *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. Doctoral Dissertation.
34. Skoglund, Alexander (2009) *Programming by Demonstration of Robot Manipulators*. Doctoral Dissertation.
35. Ranjbar, Parivash (2009) *Sensing the Environment: Development of Monitoring Aids for Persons with Profound Deafness or Deafblindness*. Doctoral Dissertation.
36. Magnusson, Martin (2009) *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. Doctoral Dissertation.
37. Rahayem, Mohamed (2010) *Segmentation and fitting for Geometric Reverse Engineering. Processing data captured by a laser profile scanner mounted on an industrial robot*. Doctoral Dissertation.
38. Karlsson, Alexander (2010) *Evaluating Credal Set Theory as a Belief Framework in High-Level Information Fusion for Automated Decision-Making*. Doctoral Dissertation.
39. LeBlanc, Kevin (2010) *Cooperative Anchoring – Sharing Information About Objects in Multi-Robot Systems*. Doctoral Dissertation.
40. Johansson, Fredrik (2010) *Evaluating the Performance of TEWA Systems*. Doctoral Dissertation.
41. Trincavelli, Marco (2010) *Gas Discrimination for Mobile Robots*. Doctoral Dissertation.
42. Cirillo, Marcello (2010) *Planning in Inhabited Environments: Human-Aware Task Planning and Activity Recognition*. Doctoral Dissertation.
43. Nilsson, Maria (2010) *Capturing Semi-Automated Decision Making: The Methodology of CASADEMA*. Doctoral Dissertation.
44. Dahlbom, Anders (2011) *Petri nets for Situation Recognition*. Doctoral Dissertation.
45. Ahmed, Muhammad Rehan (2011) *Compliance Control of Robot Manipulator for Safe Physical Human Robot Interaction*. Doctoral Dissertation.
46. Riveiro, Maria (2011) *Visual Analytics for Maritime Anomaly Detection*. Doctoral Dissertation.

47. Rashid, Md. Jayedur (2011) *Extending a Networked Robot System to Include Humans, Tiny Devices, and Everyday Objects*. Doctoral Dissertation.
48. Zain-ul-Abdin (2011) *Programming of Coarse-Grained Reconfigurable Architectures*. Doctoral Dissertation.
49. Wang, Yan (2011) *A Domain-Specific Language for Protocol Stack Implementation in Embedded Systems*. Doctoral Dissertation.
50. Brax, Christoffer (2011) *Anomaly Detection in the Surveillance Domain*. Doctoral Dissertation.
51. Larsson, Johan (2011) *Unmanned Operation of Load-Haul-Dump Vehicles in Mining Environments*. Doctoral Dissertation.
52. Lidström, Kristoffer (2012) *Situation-Aware Vehicles: Supporting the Next Generation of Cooperative Traffic Systems*. Doctoral Dissertation.
53. Johansson, Daniel (2012) *Convergence in Mixed Reality-Virtuality Environments. Facilitating Natural User Behavior*. Doctoral Dissertation.
54. Stoyanov, Todor Dimitrov (2012) *Reliable Autonomous Navigation in Semi-Structured Environments using the Three-Dimensional Normal Distributions Transform (3D-NDT)*. Doctoral Dissertation.
55. Daoutis, Marios (2013) *Knowledge Based Perceptual Anchoring: Grounding percepts to concepts in cognitive robots*. Doctoral Dissertation.
56. Kristoffersson, Annica (2013) *Measuring the Quality of Interaction in Mobile Robotic Telepresence Systems using Presence, Spatial Formations and Sociometry*. Doctoral Dissertation.
57. Memedi, Mevludin (2014) *Mobile systems for monitoring Parkinson's disease*. Doctoral Dissertation.
58. König, Rikard (2014) *Enhancing Genetic Programming for Predictive Modeling*. Doctoral Dissertation.
59. Erlandsson, Tina (2014) *A Combat Survivability Model for Evaluating Air Mission Routes in Future Decision Support Systems*. Doctoral Dissertation.
60. Helldin, Tove (2014) *Transparency for Future Semi-Automated Systems. Effects of transparency on operator performance, workload and trust*. Doctoral Dissertation.

61. Krug, Robert (2014) *Optimization-based Robot Grasp Synthesis and Motion Control*. Doctoral Dissertation.
62. Reggente, Matteo (2014) *Statistical Gas Distribution Modelling for Mobile Robot Applications*. Doctoral Dissertation.
63. Längkvist, Martin (2014) *Modeling Time-Series with Deep Networks*. Doctoral Dissertation.
64. Hernández Bennetts, Víctor Manuel (2015) *Mobile Robots with In-Situ and Remote Sensors for Real World Gas Distribution Modelling*. Doctoral Dissertation.
65. Alirezaie, Marjan (2015) *Bridging the Semantic Gap between Sensor Data and Ontological Knowledge*. Doctoral Dissertation.
66. Pashami, Sepideh (2015) *Change Detection in Metal Oxide Gas Sensor Signals for Open Sampling Systems*. Doctoral Dissertation.
67. Lagriffoul, Fabien (2016) *Combining Task and Motion Planning*. Doctoral Dissertation.
68. Mosberger, Rafael (2016) *Vision-based Human Detection from Mobile Machinery in Industrial Environments*. Doctoral Dissertation.
69. Mansouri, Masoumeh (2016) *A Constraint-Based Approach for Hybrid Reasoning in Robotics*. Doctoral Dissertation.
70. Albitar, Houssam (2016) *Enabling a Robot for Underwater Surface Cleaning*. Doctoral Dissertation.
71. Mojtahedzadeh, Rasoul (2016) *Safe Robotic Manipulation to Extract Objects from Piles: From 3D Perception to Object Selection*. Doctoral Dissertation.
72. Köckemann, Uwe (2016) *Constraint-based Methods for Human-aware Planning*. Doctoral Dissertation.
73. Jansson, Anton (2016) *Only a Shadow. Industrial Computed Tomography Investigation, and Method Development, Concerning Complex Material Systems*. Licentiate Thesis.
74. Sebastian Hällgren (2016) *Valuable shapes from powdery materials – From small beginnings come great things; When and How to design for Additive Manufacturing*. Licentiate Thesis.
75. Junges, Robert (2017) *A Learning-driven Approach for Behavior Modeling in Agent-based Simulation*. Doctoral Dissertation.

76. Ricão Canelhas, Daniel (2017) *Truncated Signed Distance Fields Applied To Robotics*. Doctoral Dissertation.