TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Computer Aided Medical Procedures & Augmented Reality / I16

Reconstruction and Scalable Detection and Tracking of 3D Objects

Wadim Kehl

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

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Daniel Cremers
Prüfer der Dissertation:	
1.	Prof. Dr. Nassir Navab
2.	Prof. Dr. Horst Bischof
3.	Prof. Dr. Matthias Nießner

Die Dissertation wurde am 25.09.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 30.01.2020 angenommen.

Abstract

The task of detecting objects in images is essential for autonomous systems to categorize, comprehend and eventually navigate or manipulate its environment. Since many applications demand not only detection of objects but also the estimation of their exact poses, 3D CAD models can prove helpful since they provide means for feature extraction and hypothesis refinement. This work, therefore, explores two paths: firstly, we will look into methods to create richly-textured and geometrically accurate models of real-life objects. Using these reconstructions as a basis, we will investigate on how to improve in the domain of 3D object detection and pose estimation, focusing especially on scalability, i.e. the problem of dealing with multiple objects simultaneously.

A fundamental aspect of the thesis is the usage of RGB-D sensors to tackle the above-mentioned tasks. In contrast to standard color cameras, these sensors provide an additional depth channel that supplies to each pixel the metric distance to the camera. This allows for correct depth perception and alleviates many typical problems such as scale estimation and occlusion reasoning.

The part on reconstruction will start with a method that allows for recovering the full colored geometry of arbitrarily shaped objects. By tracking the camera movement via the object's support surface, we can eventually fuse keyframes into a colored signed distance field after global pose optimization. By repositioning the object to expose unseen parts, we create multiple partial scans and propose a novel variational fusion scheme. The reconstruction quality exhibited supersedes related methods and allows for metrically accurate models. In a follow-up work, we focus on a more efficient method to achieve the fusion by means of Octrees. These spatial look-up structures allow for memory-efficient storage but are not straightforward to use in optimization tasks.

The part on detection will first focus on hashing of templated object views to achieve scalability. While discriminative, most template approaches suffer from a linear time complexity since each template has to be matched against the scene. With our learned hashing scheme, we decrease the computational complexity with only a small penalty on the detection accuracy. From there, we present our second work in that domain that employs Deep Learning of local RGB-D patches to allow for robust voting of object instances. This method is scalable and performs well in cluttered scenes at reasonable speeds. Following up, we introduce a novel deeply-learned detection scheme that predicts 2D bounding boxes together with scored 6D poses in a single shot. Our approach scales well to many objects and can run at 10Hz. Lastly, we present a model tracker in RGB-D data by means of a direct energy minimization over contour and object-interior cues. Our elegant method is robust to occlusion and scale changes and runs on a single CPU core for multiple objects in real-time.

Zusammenfassung

Objekterkennung in Bildern ist für ein autonomes System von entscheidender Bedeutung, um seine Umgebung zu kategorisieren, zu erfassen und schließlich zu navigieren oder zu manipulieren. Da viele Anwendungen nicht nur die Erkennung von Objekten, sondern auch die Schätzung ihrer exakten Positionen erfordern, können sich 3D-CAD-Modelle als hilfreich erweisen, da sie Mittel zur Merkmalsextraktion und Verfeinerung von Hypothesen bereitstellen. In dieser Arbeit werden daher zwei Wege untersucht: Erstens werden wir Methoden untersuchen, um strukturreiche und geometrisch genaue Modelle realer Objekte zu erstellen. Auf der Grundlage dieser Konstruktionen werden wir untersuchen, wie sich der Bereich der 3D-Objekterkennung und der Posenschätzung verbessern lässt, wobei insbesondere die Skalierbarkeit im Vordergrund steht, d.h. das Problem der gleichzeitigen Bearbeitung mehrerer Objekte.

Ein grundlegender Aspekt der Arbeit sind RGB-D-Sensoren zur Bewältigung der oben genannten Aufgaben. Im Gegensatz zu Standard-Farbkameras bieten diese Sensoren einen zusätzlichen Tiefenkanal, der jedem Pixel den metrischen Abstand zur Kamera liefert. Dies ermöglicht eine korrekte Tiefenwahrnehmung und mindert viele typische Probleme wie Skalenschätzung und Verdeckung.

Der Teil der Rekonstruktion beginnt mit einer Methode, mit der die vollfarbige Geometrie beliebig geformter Objekte erfasst werden kann. Indem wir die Kamerabewegung über die Oberfläche des Objekts verfolgen, können wir nach der globalen Posenoptimierung die Keyframes zu einem farbigen, vorzeichenbehafteten Distanzfeld zusammenfügen. Indem wir das Objekt neu positionieren um unsichtbare Teile freizulegen, erstellen wir mehrere Teilscans und schlagen ein neues Variationsfusionsschema vor. Die gezeigte Rekonstruktionsqualität ersetzt verwandte Methoden und ermöglicht metrisch genaue Modelle. In einer weiteren Arbeit konzentrieren wir uns auf eine effizientere Methode, um die Fusion mittels Octrees zu erreichen. Diese räumlichen 'Look-Up'-Strukturen ermöglichen eine speichereffiziente Speicherung, sind jedoch für Optimierungsaufgaben nicht einfach zu verwenden.

Der Teil der Erkennung konzentriert sich zunächst auf das Hashing von Objektansichten als Templates für Skalierbarkeit. Obwohl diskriminierend, leiden die meisten Template-Ansätze unter einer linearen Zeitkomplexität, da jedes Template mit der Szene verglichen werden muss. Mit unserem gelernten Hashing-Schema verringern wir die Rechenkomplexität mit nur einem kleinen Verlust in Erkennungsgenauigkeit. Darauffolgend präsentieren wir unsere zweite Arbeit in diesem Bereich, welche Deep Learning auf lokale RGB-D-Patches anwendet, um ein zuverlässiges Matching von Objektinstanzen zu ermöglichen. Diese Methode ist skalierbar und eignet sich für überfüllte Szenen mit angemessener Geschwindigkeit. Im Anschluss daran stellen wir ein neuartiges Erkennungsschema mittels Deep Learning vor, welches 2D-Bounding-Boxen zusammen mit 6D-Posen in einem einzigen Inferenzschritt liefert. Unser Ansatz ist für viele Objekte gut skalierbar und kann mit 10 Hz ausgeführt werden. Zuletzt präsentieren wir einen Modell-Tracker in RGB-D-Daten mittels direkter

Energieminimierung über Objektkontur und Innenfläche. Unsere elegante Methode ist robust gegen Verdecking und Skalenänderungen und läuft für mehrere Objekte in Echtzeit auf einem einzigen CPU-Kern.

Acknowledgments

Overall, I spent four wonderful years at the Computer Aided Medical Procedures (CAMP) group. My first wave of gratitude goes to Prof. Dr. Nassir Navab, PD Dr. Federico Tombari and PD Dr. Slobodan Ilic. It was this trio that kept me going forwards and pulling me up when I was stumbling over all the stepping stones of the "PhD journey"^{TM}. I am convinced that I would not have reached the summit of the doctoral trail without the influence of every single piece of that trinity.

I have interacted with many CAMPers at that time which grew fond to my heart! Fausto, who started out as an office mate and kept following me around ever since; Vasileios (Walter), who was there from the start, origin unknown; Mira, who decided to let me supervise her Master's thesis out of sheer desperation and went further from there, followed by Fabian, Sergey and Adrian, which all decided to share the same fate as her; and finally, Tolga the Beast, who was churning out countless publications.

My time at CAMP was a funny, lovely, blurry mess of learning and teaching, of laughter and tears. I have shared that with numerous people that walked along the same path and I want to thank everyone that was there to share the ride. A honorable mention goes out the Hilla family as well which kept the CAMP group in a running state during the whole time.

A special 'thank you' goes out to Toyota Europe for funding and supporting my research throughout the whole time. Kobori-san, Anja-San, Gianpiero-san and Sven-san have given me a lot of support and trust and I am glad that I was able to repay their trust with my work. And thanks for all the Japan trips; I loved them!

I also want to thank Prof. Dr. Matthias Nießner and Prof. Dr. Horst Bischof for reviewing my work!

Last but not least, I am thankful to my parents Jakob and Alla, my sisters Viktoria and Nicole and my brother Andreas, as well as my wife Antonina for all the support over the years of my life.

-Wadim

When you're not concerned with succeeding, you can work with complete freedom.
Larry David.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer vision has developed into an essential component of modern-day systems and its application in domains such as logistics, surveillance, entertainment, medicine, social media, robotics or slowly also autonomous driving, is ubiquitous. As a research field, computer vision has gained much traction in the last two decades because of more robust algorithms and faster computers, enabling practical usage.

At its core, computer vision deals with the analysis of images with the goal of inverting the imaging process. Behind every (natural) 2D image, there is a 3D world that explains that image. Formally, given a world state $\mathbb{W}$ and a camera $\mathbb{C}$, there exists an imaging process

$$f : \mathbb{C} \times \mathbb{W} \to RGB \tag{1.1}$$

that reduces the world to pixels. Let us assume that, for the sake of argument, we could invert the imaging function f s.t.

$$f^{-1} : RGB \to \mathbb{C} \times \mathbb{W} \tag{1.2}$$

would allow us to retrieve the state of the world and the camera from a single image. Obviously, this formalism is hopelessly abstract since it would require proper models for the world and the camera. And even if we were able to write down f correctly, it is known that converting two-dimensional imagery to three-dimensional geometry is ill-posed and non-bijective. There are infinitely many instances of 3D worlds that can be represented by the same 2D image.

But what if we were able to crudely approximate the world state by breaking it apart into smaller, more tractable pieces? We could find a decomposition

$$\mathbb{W} \sim \mathbb{W}^* := \mathbb{O} \times \mathbb{I} \times \mathbb{G} \times \mathbb{M} \times \cdots \tag{1.3}$$

that allows to (partially) express the world in terms of objects, illumination, geometry, materials and so forth. This is, in fact, the neighboring domain of computer graphics where synthetic worlds are constructed via such tractable pieces and then rendered, mimicking the imaging function f.

Our goal in computer vision, though, is the recovery of the world via the inverse f^{-1}. We could use our world approximation $\mathbb{W}^*$ and a possible camera

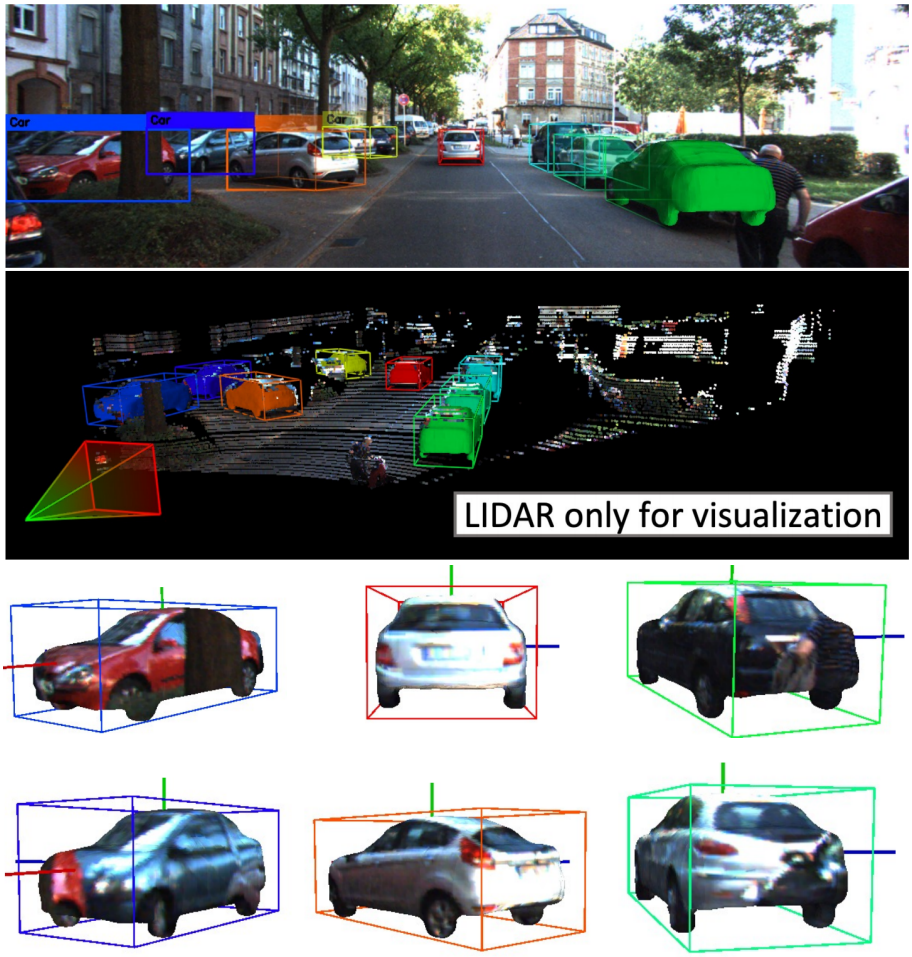


Figure 1.1: Taken from [Manhardt et al., 2019b] that shows a partial reconstruction of the world, namely of cars in the scene. Top: Detection and estimation of cars from a single image. Bottom: Recovered metric models from the image and projectively textured.

approximation (e.g. pinhole model) $\mathbb{C}^*$ to make the problem feasible. As an example of such a crude recovery of the world state and as a general motivation for this thesis, we present in Figure 1.1 an image from our publication *ROI-10D: Monocular Lifting of 2D Detection to 6D Pose and Metric Shape* that deals with detection, pose estimation and metric shape reconstruction of cars from a single image.

In this work we have shown that we can reconstruct the (partial) world state of interest by modelling the necessary components. We can recover $\mathbb{W}^*$ via constraining the possible objects' shapes and poses, as well as inducing metric priors that allow to estimate the proper scale of the world. This example shows us that we do not need to reconstruct the world in its entirety, but can instead focus on an approximation of a subset of the world to devise useful applications.

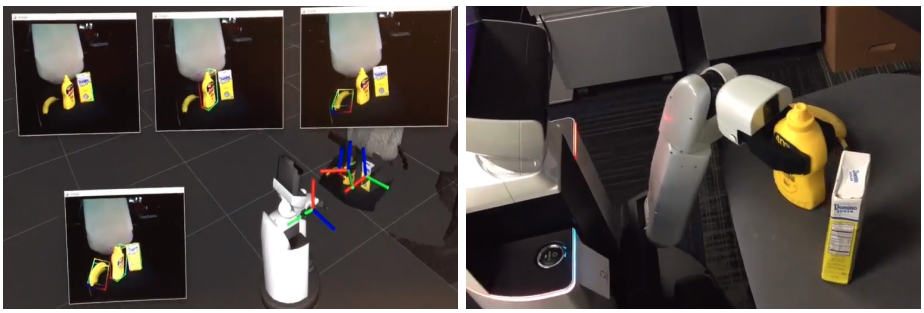


Figure 1.2: A robotic example depicting our pipeline for object detection, pose estimation and subsequent pose refinement, enabling robotic manipulation and general interaction with the environment.

In this specific case, our detection and metric reconstruction would allow a self-driving car to identify other agents in the scene and to navigate around those entities. Provided with proper priors, we can even undo the imaging process and can recover the absolute (metric) scale of the world which, as we already mentioned, is ill-posed in general.

Another example that motivates this thesis is shown in Figure 1.2 where two of our publications, *SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again*, and *Deep Model-Based 6D Pose Refinement in RGB*, are used in conjunction to build a pipeline for monocular object detection, pose estimation and continuous pose refinement. The pipeline is trained on a set of provided, previously reconstructed 3D models, and allows a household robot to identify and manipulate objects of interest in the environment.

1.1 Contributions

In this work, we will touch upon a series of domains of computer vision: 3D reconstruction of objects, their detection and estimation of their poses as well as tracking their poses over time. Although these domains can stand on their own and do not necessarily overlap, they all contributed to the final culmination of the publications mentioned above. The main contributions of the thesis are summarized as follows:

- We investigate the problem of 3D metric object reconstruction. Provided with RGB-D imagery, our goal is to recover a colored 3D mesh of the object by means of fusing multiple partial reconstructions. Building on prior work, we present a variational optimization scheme over multiple signed distance fields that produces models of high fidelity. Following that, we devise a second method that uses Octrees for a more efficient fusion while retaining overall reconstruction quality.

- For object detection and pose estimation, we present a method towards learning decision trees on hashed object view templates. By respecting the relation of spatially neighboring views, we propose a learning scheme

that enables higher recall rates than a purely feature-driven approach and scales well with many views and objects.

- Following the topic, we will present approaches based on Deep Learning. The first approach introduces the notion of local 6D pose voting and a method to learn descriptors from local RGB-D image patches. These descriptors are then used for scalable matching of objects to the scene to create hypotheses with a high recall. The second approach takes a more holistic perspective and uses a single-shot detection network to simultaneously detect objects and their poses. The approach is able to run in real-time and produces accurate predictions using only RGB information.

- The last contribution of this thesis is a scalable 6D pose tracker that is able to run in real-time for multiple objects on a single CPU core. We propose to fuse color and depth information into a joint framework for pose optimization and provide approximations that enable real-time tracking without GPU involvement.

Taken together, the contributions of the thesis allow for an end-to-end pipeline for the generation of 3D object models from the environment, and the subsequent detection and tracking of these objects in the wild. In essence, this thesis could be taken as an immediate guideline for the creation of augmented reality software or industrial verification applications.

The mentioned contributions will be laid out in more detail in the following chapters.

1.2 Thesis Outline

We provide an overview for each chapter of the thesis. All of the presented methods have been published at major peer-reviewed, double-blind conferences and we therefore additionally provide the related work at the beginning of each part of the thesis.

Chapter 2. We present the theoretical background of this work. In particular, we review affine and projective geometry, rigid-body transformations, signed distance fields as well as convolutional neural networks.

Part I: Accurate Reconstruction of 3D Objects

Chapter 3. We will start with object reconstruction via RGB-D sensors and volumetric distance (SDF) representations. To this end, we introduce our framework that uses a tabletop setup and multiple scan iterations to recover the full colored geometry by dense alignment.

Chapter 4. Building on the previous chapter, we present a novel formulation for geometric fusion of multiple SDFs with the help of a dynamic Octree structure. This optimization scheme drastically reduces the memory footprint while retaining geometric fidelity.

Part II. Scalable Detection and Tracking of 3D Objects

Chapter 5. Our first work on object detection deals with learned hashing of multi-modal viewpoint descriptors for scalable object retrieval with a sliding window. We will show that properly learning the hash bucket distributions leads to better detection and 6D pose recall rates.

Chapter 6. Differently to hashing, this chapter will explore deep learning of local RGB-D patches with an auto-encoder network for nearest-neighbor retrieval and 6D vote casting in the scene. A final filtering and verification step ensures that our high recall rate is coupled with high precision.

Chapter 7. Orthogonally to the previous approaches, this chapter introduces the idea of single-shot detection (SSD) for 6D pose estimation. A fully-convolutional network learns on synthetic data to classify and predict bounding boxes as well as viewpoint information. These predictions can then be composed into 6D poses and further refined to accurate pose estimates.

Chapter 8. As a final work, we introduce a lightweight, CPU-based method for model-based 6D pose tracking in color and depth data. With the help of approximations and 3D prerenderings, the method achieves real-time tracking of multiple instances under occlusion and drastic scale changes.

Chapter 9. This chapter will conclude the thesis with remarks on the achieved results, their limitations and possible future directions.

Chapter 2

Background

We will first introduce basic theory on algebraic representations and transformations since these form an essential part of all presented methods. Furthermore, this chapter will help in establishing a common notation for all needed mathematical entities and explain on how to conduct optimization over rigid-body motions. After that, we will focus on the concept of signed distance fields since both parts of the thesis rely on it. Lastly, we will give a brief introduction on Deep Learning to render the final chapters easier to understand.

2.1 Representations and Transformations

Since the domain of Euclidean and projective geometry is vast, we will focus solely on those parts that are essential for understanding the theory in this thesis. For more explanations we refer the reader to [Hartley and Zisserman, 2004].

In general, we will denote points in 2D as $\mathbf{x} = (x, y)^\top \in \mathbb{R}^2$ and points in 3D as $\mathbf{X} = (X, Y, Z)^\top \in \mathbb{R}^3$. Given this definition, we can introduce the notion of homogeneous coordinates where we embed those points into projective spaces $\tilde{\mathbf{x}} = (x, y, w)^\top \in \mathbb{P}^2$ and $\tilde{\mathbf{X}} = (X, Y, Z, W)^\top \in \mathbb{P}^3$. Such a representation has two advantages: it allows us to easily express projective relations and to write rigid-body (affine) motions as linear transformations via matrix-vector products.

Provided a 3D point $\mathbf{X}$ with the homogeneous part set to 1 ($W = 1$), we can write the rotation with $\mathbf{R} \in \mathrm{SO}(3) \subset \mathbb{R}^{3\times3}$ and translation with $\mathbf{T} \in \mathbb{R}^3$ as

$$\begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{pmatrix} \cdot \tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} \cdot \mathbf{X} + \mathbf{T} \\ 1 \end{pmatrix}. \tag{2.1}$$

2.1.1 Rotations in 3D Space

While translations are simply added to a point, rotations are more complex and must satisfy certain properties. The special orthogonal group SO(3) represents the space of all 3D rotations, consisting of matrices $\mathbf{R} \subset \mathbb{R}^{3\times3}$ that satisfy the

two conditions

$$\mathbf{R}^\top \cdot \mathbf{R} = \mathbf{R} \cdot \mathbf{R}^\top = \mathbf{I} \qquad \text{and} \qquad \det(\mathbf{R}) = 1. \tag{2.2}$$

From those two constraints, it is evident that all elements in this group preserve both local orientation and length. Note that rotations, and therefore also rigid-body motions, are not commutative and a change in application order changes the final transformation. The parametrization with $3 \cdot 3 = 9$ values allows for easy application to points in linear form but is not intuitive since rotations in 3D have only 3 degrees of freedom (DoF).

There exist multiple ways to represent rotations in 3D space, although the simplest and most used representation are Euler angles α, β, γ, representing the amount of rotation around each canonical axis from which a final rotation matrix can be constructed. Euler angles are the most compact representation but they are not unique, meaning different angles can result in the same final rotation matrix. Additionally, the Euler representation can suffer from the gimbal lock problem, leading to the loss of one degree of freedom for a certain order of rotation application.

Unit Quaternion

Another viable representation for rotations are unit quaternions. They can be seen as an extension of complex numbers, comprise 4 values, $\mathbf{q} = (q_w, q_x, q_y, q_z)^\top$ with unit norm $||\mathbf{q}|| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1$ and inverse element $\mathbf{q}^{-1} = \bar{\mathbf{q}} = (q_w, -q_x, -q_y, -q_z)$ as the conjugate. In fact, this representation is closely related to the axis-angle formulation since the angle of rotation is encoded in the real part, $q_w = \cos\theta/2$, whereas the imaginary part serves as the normalized rotation axis $(q_x, q_y, q_z) = \overrightarrow{\mathbf{r}} \sin\theta/2$.

Unit quaternions are widely-used in vision and robotics for their compactness and clarity, and although they suffer from the minor problem of a dual representation, i.e. $\mathbf{q}$ and $-\mathbf{q}$ are representing the same rotation, they usually are easy to handle in algorithms. For example, the rotation of a 3D point $\mathbf{X}$ can be accomplished by first embedding it into the imaginary part of a quaternion, $\mathbf{X_q} = (0, X, Y, Z)$, and then applying a rotation via $\mathbf{q} \cdot \mathbf{X_q} \cdot \mathbf{q}^{-1} = (0, X', Y', Z')$. This operation entails less computational complexity than rotating via matrix-vector products.

In general, it is straight-forward to switch from one representation to the other. For example, we can construct a 3×3 rotation matrix from $\mathbf{q}$ via

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 \end{pmatrix}. \tag{2.3}$$

Note that each representation has specific mechanics for rotation application and concatenation. While concatenating rotation matrices $\mathbf{R}_1, \mathbf{R}_2$ is a simple matrix product $\mathbf{R}_1 \cdot \mathbf{R}_2$, concatenating two quaternions $\mathbf{q}, \mathbf{p}$ involves conjugation $\mathbf{q} \mapsto \mathbf{q} \cdot \mathbf{p} \cdot \mathbf{q}^{-1}$, similar to above.

We will introduce another variant, namely the exponential representation, in Section 2.1.5 to motivate iterative optimization over SE(3) pose manifolds.

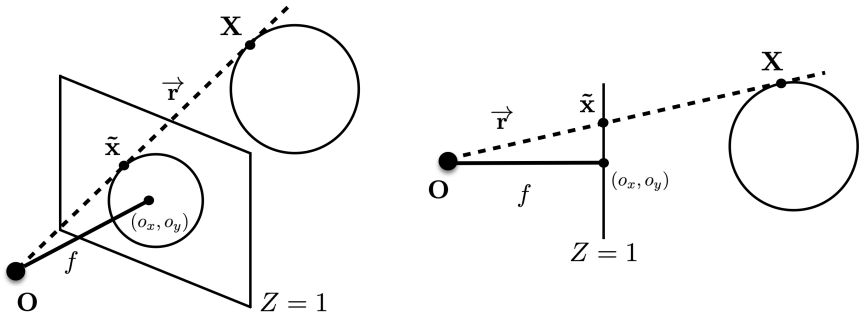


Figure 2.1: Perspective projection model from an isometric (left) and an orthogonal (right) view, parametrized via camera intrinsics. With the projection center in the world origin $\mathbf{O}$, each 2D point on the image plane $\tilde{\mathbf{x}}$ can be traced to a 3D point $\mathbf{X}$ along a projective ray. Note that the actual pinhole model has the imaging plane inverted and located behind the center of projection.

2.1.2 Pinhole Camera Model

In computer vision, a widely used projection model is the so-called pinhole camera model. It assumes a lens-free, linear projection of a 3D point onto the image plane through a single focal point. Although approximate, this model is simple to compute, precise enough for most applications and straight-forward to calibrate. In its most basic form it is parametrized by a focal length f, describing the distance between projection center and the image plane, and a principal point $(o_x, o_y)^\top$ denoting the intersection of the optical axis with the image plane. Mathematically, this projection can be written as

$$\tilde{\mathbf{x}} = \mathbf{K} \cdot \mathbf{X} = \begin{pmatrix} f & 0 & o_x \\ 0 & f & o_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.4}$$

and the result is a projective point $\tilde{\mathbf{x}} \in \mathbb{P}^2$, and in turn, the set of all points along a projective line that are mapped to the same point on the image plane. The focal length maps metric scale to pixel scale while the principal point identifies the exact imaging centre of the sensor. For convenience, we define the image plane to be the set $\{\tilde{\mathbf{x}} \mid \tilde{\mathbf{x}} = (x, y, 1)^\top\}$ of all homogeneous points with the last component set to 1. Throughout the thesis, we will use a projection function $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ that maps a 3D point to its unique 2D image plane projection $\mathbf{x} = (x, y)^\top$, coinciding with $\tilde{\mathbf{x}} = (x, y, 1)$:

$$\pi(\mathbf{X}) := \left(\frac{f \cdot X}{Z} + o_x \ , \ \frac{f \cdot Y}{Z} + o_y \right)^\top . \tag{2.5}$$

An illustration of the perspective projection model can be seen in Figure 2.1. Many problems often also require the inverse path, i.e. reprojecting a 2D point back to its 3D position. As mentioned, the inversion is not uniquely defined since infinitely many 3D points collapse into the projected 2D position. Nonetheless, the inverse projection matrix $\mathbf{K}^{-1}$ allows to compute the metric projective ray

$\overrightarrow{\mathbf{r}}$ (also called bearing vector) back into the world for any given pixel. This means that $\mathbf{K}^{-1} \cdot \tilde{\mathbf{x}} = \overrightarrow{\mathbf{r}}$ spans a one-dimensional space through the camera origin that includes the original 3D point. Additionally, if the depth Z is known or can be found (e.g. by triangulating bearing vectors from multiple cameras observing the same 2D point), the 3D point can be recovered as $\mathbf{X} = \overrightarrow{\mathbf{r}} \cdot Z$.

2.1.3 RGB-D data

Since some chapters in this work will rely on RGB-D sensors, we outline here the mathematical structure of their data. Let us define $\Omega \subset \mathbb{R}^2$ as the image plane (discarding from now on the homogeneous co-representation) such that we can introduce a color image $I : \Omega \to \mathbb{R}^3$ that maps a 2D position to an RGB value, and a depth image $D : \Omega \to \mathbb{R}^+$ that maps pixel position to a positive metric distance value.

Given the intrinsic parameters of our pinhole camera model together with a depth image, we can formulate (similar to the last subsection) the inverse function $\Pi_D : \Omega \to \mathbb{R}^3$ that maps a 2D image point back to its 3D scene point:

$$\Pi(\mathbf{x})_D := \left(\frac{x - o_x}{f} \cdot D(\mathbf{x}) \ , \ \frac{y - o_y}{f} \cdot D(\mathbf{x}) \ , \ D(\mathbf{x}) \right)^\top . \tag{2.6}$$

As mentioned before, we call this a *reprojection* since it undoes the loss of absolute scale. We can from this build a representation, sometimes referred to as an organized point cloud, where the reprojected 3D points are arranged in the same 2D lattice. This allows for straightforward indexing and (projective) nearest-neighbor searches. A visualization of the sensor data and the induced colored point cloud can be seen in Figure 2.2.

2.1.4 Iterative Closest Point method

We will introduce the topic of optimization in the space of rigid-body motions by first explaining the ICP algorithm. The Iterative Closest Point (ICP) method is a very popular scheme for aligning two point cloud sets. Although there have been many alterations and additions to the original version, we will focus here only on the two most fundamental variants: the point-to-point energy formulation from [Besl and McKay, 1992] as well as the point-to-plane energy from [Chen and Medioni, 1991].

Formally, we seek the alignment of a 3D source point set $\mathcal{S} := \{\mathbf{S}_1, ..., \mathbf{S}_n\}$ to a destination point set $\mathcal{D} := \{\mathbf{D}_1, ..., \mathbf{D}_m\}$. We further restrain ourselves to rigid-body motions, i.e. we are interested in solving for 3D rotation and 3D translation, leading to six degrees of freedom. Additionally, we assume to have been provided with direct 1-to-1 correspondences, i.e. $n = m$ and we want to minimize the distance between each pair $(\mathbf{S}_i, \mathbf{D}_i)$, leading to a general point-to-point formulation

$$\underset{\mathbf{R},\mathbf{T}}{\arg\min} \sum_i \left(||\mathbf{R} \cdot \mathbf{S}_i + \mathbf{T} - \mathbf{D}_i|| \right)^2 \quad s.t. \quad \mathbf{R} \in \mathrm{SO}(3). \tag{2.7}$$

The energy minimizes a sum of squared residuals and the authors presented a closed-form solution to the problem. Although the error function seems simple,

Figure 2.2: A frame from the LineMOD dataset [Hinterstoisser et al., 2012a]. The top row shows a registered image pair of color and depth. The depth channel is presented as a heatmap for better visualization. The image at the bottom shows the colored point cloud data produced from the application of Eq. 2.6 and slight rotation for easier interpretation. Such data allows for better reasoning about relations in metric scale, occlusions and more generally, whether space is occupied or not. Additionally, it enables reliable estimation of scene normals and surface curvature.

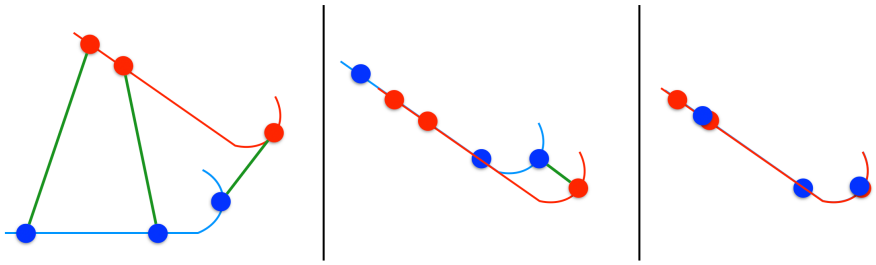


Figure 2.3: Schematic behavior of the two different ICP energies. Left: We sample points on the source surface (blue) and the destination (red) and set them in correspondence. Center: Although the point-to-point energy finds the solution with smallest point-wise differences, the actual surface alignment is not optimal. Right: The point-to-plane formulation does not penalize the sliding of the blue points along the red tangent line, resulting in better alignment overall.

the main difficulty arises in retrieving a valid rotation matrix by Singular Value Decomposition. In practice, the provided correspondence pairs $(\mathbf{S}_i, \mathbf{D}_i)$ are noisy and the energy is solved in multiple iterations with recomputed pairs until convergence.

Alternatively, if we are provided with an additional surface 3D normal $\mathbf{N}_i$ at each $\mathbf{D}_i$, a more robust point-to-plane energy can be formulated as

$$\arg\min_{\mathbf{R},\mathbf{T}} \sum_i \Big((\mathbf{R} \cdot \mathbf{S}_i + \mathbf{T} - \mathbf{D}_i) \bullet \mathbf{N}_i \Big)^2 \quad s.t. \quad \mathbf{R} \in \mathrm{SO}(3). \tag{2.8}$$

This energy allows the source points to move in the tangent planes spanned at each destination point and it can help in certain configurations to find a better fitting solution. Figure 2.3 depicts the typical convergence behavior. As before, the method is run with multiple iterations but since no closed form has been formulated for the solution, the problem is linearized and solved in succession.

Levenberg-Marquardt ICP

We will now focus our attention to a generalized iterative method, first presented in [Fitzgibbon, 2001] as LM-ICP. Let us define the alignment error as

$$E(g) := \sum_i^N ||d(g(\mathbf{S}_i), \mathbf{D}_i)||^2 \tag{2.9}$$

where g is the current transformation and d a differentiable distance measure. By defining $e_i := d(g(\mathbf{S}_i), \mathbf{D}_i) \in \mathbb{R}^3$ as the residual for correspondence i and rewriting the squared L^2 norm via dot products, we arrive at

$$E(g) := \sum_i^N e_i^\top e_i \quad \Leftrightarrow \quad E(g) := e_i^\top e_i. \tag{2.10}$$

The goal of LM-ICP is to find an update ∇g such that the new estimate $g_{k+1} = g_k + \nabla g$ reduces the error $E(g)$. To this end, a first-order Taylor series expansion

leads to

$$e_i(g+x) \approx e_i(g) + \nabla e_i(g)x = e_i + J_i x \tag{2.11}$$

with $J_i = \frac{\partial e_i}{\partial g} = \nabla e_i(g) \in \mathbb{R}^{3\times p}$ with p being the number of parameters that are used to represent the transformation. The final goal is to minimize $E(g=x)$ and therefore we solve for

$$\nabla_x E(g+x) = 2J^\top e + 2J^\top Jx \stackrel{!}{=} 0 \tag{2.12}$$

which finally leads to an Levenberg-Marquardt update as

$$x = -(J^\top J + \lambda I)^{-1} J^\top e \tag{2.13}$$

where λ steers between a Gauss-Newton step and a standard gradient descent step. If λ is high, we are sure to decrease the error while a smaller λ can lead to faster convergence if already close to the optimum.

2.1.5 Optimization over the Lie Group SE(3)

Although the concept of Lie algebras is more general and can be applied to certain vector spaces, we will focus our attention to the special Euclidean group SE(3) only. Our goal is to numerically optimize over the space of rigid-body motions which are manifolds in higher dimensions, but using a minimal representation that encode the six degrees of freedom. As mentioned before, rotation matrices $\mathbf{R} \in \mathrm{SO}(3) \subset \mathbb{R}^{3\times 3}$ satisfy certain constraints and optimizing over this manifold space requires special care. Importantly, simple linear interpolation between rotation matrices is not possible since the sum might leave the correct manifold and thus would need reprojection. If we also want to include translations, we are optimizing over a 6-dimensional manifold SE(3). Each element $\Xi \in \mathrm{SE}(3) \subset \mathbb{R}^{4\times 4}$ is a transformation matrix and has a corresponding twist $\hat{\xi} \in \mathrm{se}(3)$ from its Lie algebra, which can in turn be represented with a vector $\xi = (\mathbf{v}, \omega) = (v_1, v_2, v_3, \omega_1, \omega_2, \omega_3)^\top \in \mathbb{R}^6$, also called exponential coordinates. These coordinates themselves have a component $\mathbf{v}$ for the translation as well as ω for the rotation. Importantly, together they span the space of the se(3) algebra by means of generator matrices:

$$\hat{\xi} := v_1\mathbf{G}_1 + v_2\mathbf{G}_1 + v_3\mathbf{G}_3 + \omega_1\mathbf{G}_4 + \omega_2\mathbf{G}_5 + \omega_3\mathbf{G}_6. \tag{2.14}$$

In turn, these 4×4 generators have simple forms:

$$G_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$G_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_5 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_6 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Mapping functions between SE(3) and its Lie algebra se(3) are defined as exponential maps and logarithmic maps. Since our interest lies in optimizing well-behaving, minimal representations we focus on the exponential map

$$\exp(\hat{\xi}) := \exp\left(\begin{bmatrix}[\omega]_\times & \mathbf{v}\\ \mathbf{0} & 0\end{bmatrix}\right) = \begin{pmatrix}\mathbf{R} & \mathbf{T}\\ \mathbf{0} & 1\end{pmatrix} \tag{2.15}$$

that brings a 4×4 twist $\hat{\xi}$, generated from its 6-D coordinates vector ξ, into a proper rigid-body motion with a closed-form solution. Note that $[\omega]_\times$ represents the skew-symmetric matrix of ω.

Given a function f that we want to derive in respect to a 6 DoF transformation vector ξ at point $\mathbf{X}$, we rewrite the transformation via the generators applied to the homogeneous point $\tilde{\mathbf{X}}$:

$$\begin{aligned} f(\mathbf{X}) = \begin{pmatrix}\mathbf{R} & \mathbf{T}\\ \mathbf{0} & 1\end{pmatrix} \cdot \tilde{\mathbf{X}} =& (v_1\mathbf{G}_1 + v_2\mathbf{G}_1 + v_3\mathbf{G}_3 + \omega_1\mathbf{G}_4 + \omega_2\mathbf{G}_5 + \omega_3\mathbf{G}_6) \cdot \tilde{\mathbf{X}} \\ &= \begin{pmatrix} v_1 + \omega_2 Z - \omega_3 Y \\ v_2 - \omega_1 Z + \omega_3 X \\ v_3 + \omega_1 Y - \omega_2 X \end{pmatrix}. \end{aligned}$$

We can now see how the individual components of the rigid-body motion act simultaneously on the point. Taking the derivative in respect to each component yields a twist Jacobian of the form

$$J = \frac{\partial f}{\partial \xi} = \begin{bmatrix} \frac{\partial f_1}{\partial \xi_1} & \cdots & \cdots & \frac{\partial f_1}{\partial \xi_6} \\ \vdots & & & \vdots \\ \frac{\partial f_3}{\partial \xi_1} & \cdots & \cdots & \frac{\partial f_3}{\partial \xi_6} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & Z & -Y \\ 0 & 1 & 0 & -Z & 0 & X \\ 0 & 0 & 1 & Y & -X & 0 \end{bmatrix}. \tag{2.16}$$

Since this equation often arises in pose optimization problems during derivation, many algorithms across the fields of vision and robotics, e.g. the presented LM-ICP, use this as a building block. After all terms have been reduced, the final twist update $\nabla\hat{\xi}$ can then be applied to the current pose estimate at time t as

$$\begin{pmatrix}\mathbf{R}_{t+1} & \mathbf{T}_{t+1}\\ \mathbf{0} & 1\end{pmatrix} = \exp(\nabla\hat{\xi}) \cdot \begin{pmatrix}\mathbf{R}_t & \mathbf{T}_t\\ \mathbf{0} & 1\end{pmatrix}. \tag{2.17}$$

Note that, unlike the rotational space, the Lie algebra represents a proper vector space with scalar multiplication. This allows addition and subtraction whose results can always be mapped back to a proper element in SE(3), enabling above update scheme.

2.2 Signed Distance Fields

An important concept that will be used throughout the thesis are signed distance fields (SDF). In essence, they simply describe spatial distances for each point in this field to the closest point of a defined set (not necessarily in the field). Although the concept can be applied to many tasks, it is very often used for implicit embeddings of surfaces. Mathematically, given a spatial domain $\Omega_n \subset$

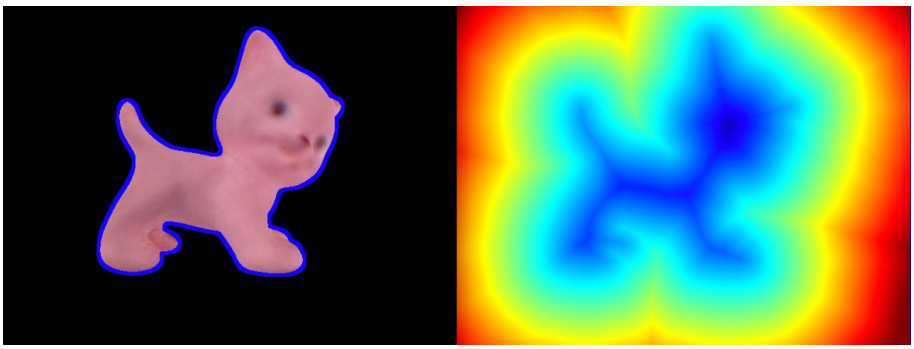


Figure 2.4: Left: A 3D model rendering together with its highlighted contour in blue. Right: A signed distance transform that gives the Euclidean distance of each pixel to the closest contour point. While the red colors are showing positive 'outside' values, the blue colors are negative 'inside' values. The teal-colored pixels are close to the boundary and therefore close to zero.

$\mathbb{R}^n$, we define a subset of 'zero-distance' points $P \subset \mathbb{R}^n$ such that a distance field $\phi^+ : \Omega_n \to \mathbb{R}^+$ is given by

$$\phi^+(\mathbf{x}) := \min_{p \in P} \mathcal{D}(\mathbf{x}, \mathbf{p}) \tag{2.18}$$

where $\mathcal{D}$ can be an arbitrary distance metric (although in practice, L^1 or L^2 are the most-widely used). Now, a surface of dimensionality $n-1$ can be represented as the zero-level set of the signed field $\phi : \Omega_n \to \mathbb{R}$, i.e. $\{\mathbf{x} \in \Omega_n \mid \phi(\mathbf{x}) = 0\}$. Having a signed field can preserve a sense of orientation and define an 'outside' and 'inside' area. This usually requires an external piece of information such as a viewpoint when encoding distances to the camera or normal orientation when embedding 3D geometry. See Figure 2.4 for a visualization.

2.2.1 Data-driven SDF creation

In the most naive approach the field can be computed exhaustively in a brute-force manner. This means that for any given binary input image, the distance value for each field element can be determined by traversing the whole image and computing the minimum over all distances from the closest zero-value points. Unfortunately, the complexity grows exponentially with the number of dimensions n, rendering this approach impractical. For certain problems, such as computing an unsigned distance for L_2 of L_1 norms, the work [Felzenszwalb and Huttenlocher, 2012] proposes to separate the problem of n-dimensional distance fields into envelopes of one-dimensional fields, allowing for linear time computation.

Another variant which we will use in this thesis (Chapters 3 and 4) are projective SDFs, pioneered by [Curless and Levoy, 1996], and used in many recent methods that deal with volumetric 3D reconstruction of partial views [Newcombe et al., 2011, Nießner et al., 2013, Slavcheva et al., 2016]. Here, the input is not a binary image but rather a set of observed 3D surfaces from a

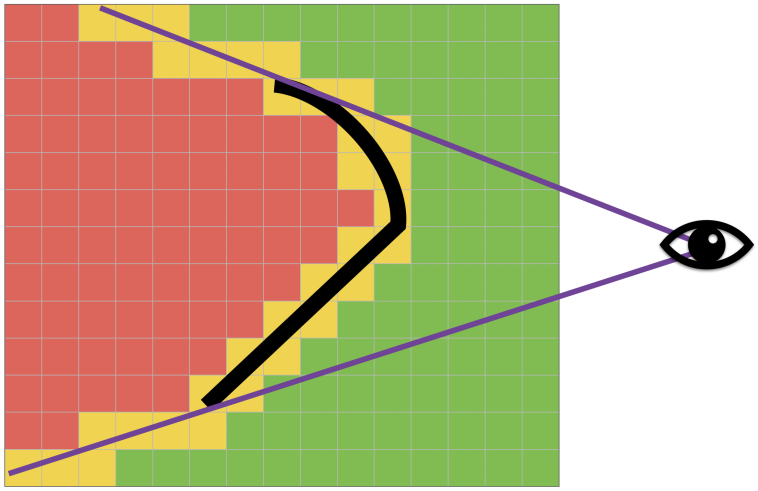


Figure 2.5: Given an observed surface (black line), each SDF position calculates a projective distance by shooting a ray towards the viewpoint and checking for obstruction. Green signifies area in front of the surface whereas red area is behind the surface. The yellow band can be regarded as an area of uncertainty. Note that we discretized the possible values into three bins for easier visualization while the actual SDF values are usually continuous in practice.

depth image. The goal is to compute the closest distance of each position in the SDF to the closest surface. Instead of finding the closest surface point in Euclidean space, the method exploits the existence of a viewpoint to efficiently compute projective distances. Here, each point in the SDF shoots a ray towards the viewpoint and checks for obstruction along the way. If no hit occurred the point is regarded as visible, otherwise it must be behind a surface. See Figure 2.5 for a visualization.

More formally, let us assume that we have a depth map D and we want to compute a (truncated) signed distance for each point $\mathbf{X} \in \Omega_3$ in a 3D volume. Without loss of generality, we define that points in front of observed surfaces receive positive values while points behind surfaces become negative. We scale with a divisor δ and truncate to $[-1, +1]$, which can be written as:

$$\phi(\mathbf{X}) = \psi(D(\pi(\mathbf{X})) - ||\mathbf{X} - \mathbf{P}_{xyz}||) \quad , \quad \psi(d) = \begin{cases} sgn(d) & \text{if } |d| > \delta \\ \frac{d}{\delta} & \text{else} \end{cases} \tag{2.19}$$

with $\mathbf{P}_{xyz}$ being the 3D viewpoint center and π our projection function. The parameter δ can be regarded as a tolerance towards measurement noise in the depth data and should be set in respect to the depth sensor's fidelity and the acceptable margin of error.

2.3 Convolutional Neural Networks

Although neural networks have been around since the 1950s [Rosenblatt, 1956] and their convolutional counterparts since the 1980s [Fukushima, 1980], their

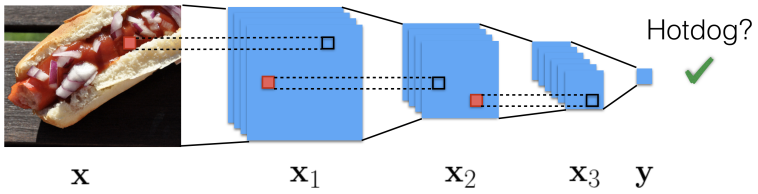


Figure 2.6: Schematic representation of a CNN hotdog classifier. Starting from input image $\mathbf{x}$, the network produces multiple intermediate feature tensors before merging into a binary classification output $\mathbf{y}$. The red boxes are strided convolution kernels that process their respective input and feed the next layer.

recent increase in popularity stems mainly from advances in parallel computing power as well as contributions in terms of architecture and mathematical operations. The machine learning community has experienced a significant change with the introduction of Deep Learning, the general term for the study of deeply-layered neural networks. The use of such models has led to tremendous improvements over existing state-of-the-art machine learning algorithms across many fields. We will focus here solely on network components used for computer vision applications.

The now famous AlexNet [Krizhevsky et al., 2012] was able to beat the (then) state-of-the-art methods by a large margin by means of a convolutional neural network (CNN) trained in parallel on two GPUs. The work introduced many additions that are still used by many methods today and, in fact, enabled CNNs to reach their unrivaled performance. We will discuss briefly the core components that constitute such CNNs.

As a minimum, a neural network f learns an input/output mapping $f(\mathbf{x}) = \mathbf{y}$ where both $\mathbf{x}, \mathbf{y}$ can be tensors of arbitrary rank or dimensionality. The mapping f itself is usually realized as a deep cascade of n differentiable layers, i.e. $f(\cdot) = f_n \circ \cdots \circ f_3 \circ f_2 \circ f_1(\cdot)$, where each layer f_i has learnable parameters θ_i. Training such networks effectively gives rise to feature hierarchies where each layer learns higher-order features conditioned on preceding layers. A visualization of a simple CNN model is depicted in Figure 2.6 where an input image $\mathbf{x}$ is reduced to a single value after multiple differentiable operations. Plugging the output $\mathbf{y}$ into a differentiable classification loss (e.g. binary cross entropy) allows for end-to-end optimization of all network parameters.

2.3.1 CNN layers

Convolutions In the context of computer vision, the input $\mathbf{x} \in \mathbb{R}^{M \times N \times C}$ is usually a multi-channel image and CNNs exploit the spatial relationship via convolutional operations. More specifically, the input $\mathbf{x}_i$ to layer i is convolved by the learnable kernel $\mathbf{k}_i$ (also called filter) and then shifted by a learnable bias $\mathbf{b}_i$. The application of layer f_i with parameters $\theta_i = (\mathbf{k}_i, \mathbf{b}_i)$ to intermediate feature tensor $\mathbf{x}_i$ can then be written as

$$\mathbf{x}_{i+1} = f_i(\mathbf{x}_i) = \mathbf{x}_i * \mathbf{k}_i + \mathbf{b}_i. \tag{2.20}$$

The non-convolutional version, also called 'fully-connected', is gained by replacing $\mathbf{k}$ with matrix $\mathbf{K}$ and the convolution with a matrix-vector product.

Activations While convolutions and fully-connected layers can be used to build operations, their linear nature reduces its usefulness since a composition of linear functions remains linear. Thus, non-linear activations are an integral part of neural networks as they enable learning arbitrarily complex mappings. In practice, most activation functions are derived from the sigmoid or ReLU (rectified linear unit) families, e.g.

$$\phi(x) = \frac{1}{1+e^{-x}} \qquad \phi(x) = \max(0, x) \tag{2.21}$$

and are used right after the linear operation to induce variance in neuron activations.

Normalization Like other machine learning methods, neural networks can be very sensitive to numerical problems since the introduced operations are unbounded in theory. This leads to unstable training where parameter updates can either vanish towards 0 or explode to high values, effectively leading to dead neurons or constantly oversatured outputs. Another issue is the so-called 'internal covariate shift' that describes the ill-posed problem of learning layers, conditioned on previous layers that undergo changes themselves. Therefore, it has become common practice to employ batch [Ioffe and Szegedy, 2015] or group normalization [Yuxin Wu, 2018]. Furthermore, convolution bias can be discarded when having a post-activation normalization after each layer since each intermediate input is whitened.

2.3.2 Training via Backpropagation

The established way to train deep networks relies on first-order optimization variants such as stochastic gradient descent (SGD) [Robbins and Monro, 1951] with momentum or adaptive variants like ADAM [Kingma and Ba, 2015], and exploiting the chain rule for back-propagation. For any given differentiable loss function $\mathcal{L}$ evaluated with network output $f(\mathbf{x})$ and expected output $\hat{\mathbf{y}}$, the gradient $\frac{\partial \mathcal{L}(f(\mathbf{x}),\hat{\mathbf{y}})}{\partial \theta}$ allows us to optimize the model parameters. Let us assume that our network consists of multiple convolution-activation blocks of the following form:

$$f_i(\mathbf{x}_i) = \phi_i(\mathbf{x}_i * \mathbf{k}_i + \mathbf{b}_i). \tag{2.22}$$

To efficiently calculate a gradient update, we can leverage the layered structure by working backwards from the output and computing the partial derivatives along the way. Starting from the last layer, we compute its gradient as

$$\frac{\partial \mathcal{L}(f(\mathbf{x}),\hat{\mathbf{y}})}{\partial \theta_n} = \mathcal{L}'(f_n(\mathbf{x}_n),\hat{\mathbf{y}}) \cdot \phi_n'(\mathbf{x}_n * \mathbf{k}_n + \mathbf{b}_n) \cdot (\mathbf{x}_n * \mathbf{k}_n + \mathbf{b}_n)'. \tag{2.23}$$

Now we can reuse this information to compute the partial derivative for the preceding layer in isolated form as

$$\frac{\partial \mathcal{L}(f(\mathbf{x}), \hat{\mathbf{y}})}{\partial \theta_{n-1}} = \frac{\partial \mathcal{L}(f(\mathbf{x}), \hat{\mathbf{y}})}{\partial \theta_n} \cdot \phi'_{n-1}(\mathbf{x}_{n-1} * \mathbf{k}_{n-1} + \mathbf{b}_{n-1}) \cdot (\mathbf{x}_{n-1} * \mathbf{k}_{n-1} + \mathbf{b}_{n-1})'. \tag{2.24}$$

Following this schema, the changes in the network can be propagated up to the first layer to assemble the full gradient and perform a descent step. Training neural networks efficiently and accurately is still ongoing research since it is difficult to capture the non-linear, highly non-convex behavior in its entirety. This renders mathematical proofs as well as proper introspection into the network challenging tasks.

Part I

Accurate Reconstruction of 3D Objects

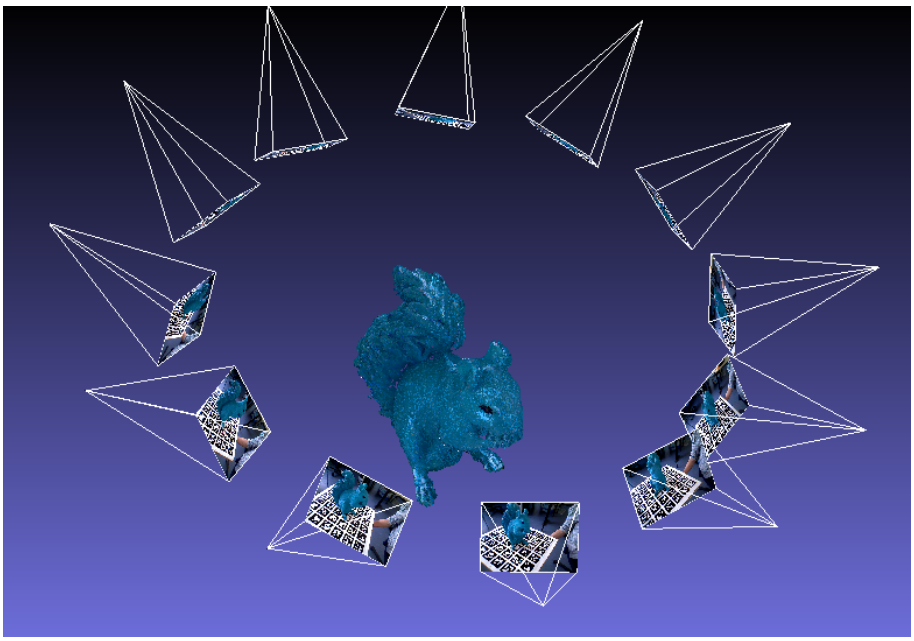


Figure 2.7: The accumulation of RGB-D point clouds from multiple viewpoints. The viewpoint poses are precise and computed from the markers beneath the object, leading to an accurate, sparse 3D reconstruction.

The first part of the thesis will deal with high-fidelity rigid 3D object reconstruction. While reconstruction itself has been the focus of computer vision for decades, the advent of commodity RGB-D sensors further widened the interest since it enabled many different user groups to create metrically accurate object reconstructions (with a good overview in [Bernardini and Rushmeier, 2002]). Reconstruction gains importance in many different fields including manufacturing verification, human entertainment like gaming or augmented reality as well as tasks in robotics such as object recognition and grasping. Proper object reconstruction from multiple views requires the knowledge of

- the (intrinsic) projection function that relates 3D geometry to 2D pixels,
- the (extrinsic) transformations that relate 3D geometry between views,
- which parts of the image depict the object in each view, and
- the means to properly fuse the data into one coherent, global frame.

One can usually roughly divide the literature up into a stationary setup, where the object sits on top of a (rotating) support surface and a dynamic setup, where the object of interest is freely moving. In Figure 2.7 we see an example of one such stationary reconstruction. Provided a planar markerboard and properly calibrated camera intrinsics, we can easily compute the precise pose of each viewpoint from the fiducials. Using a form of segmentation in either 2D (e.g.

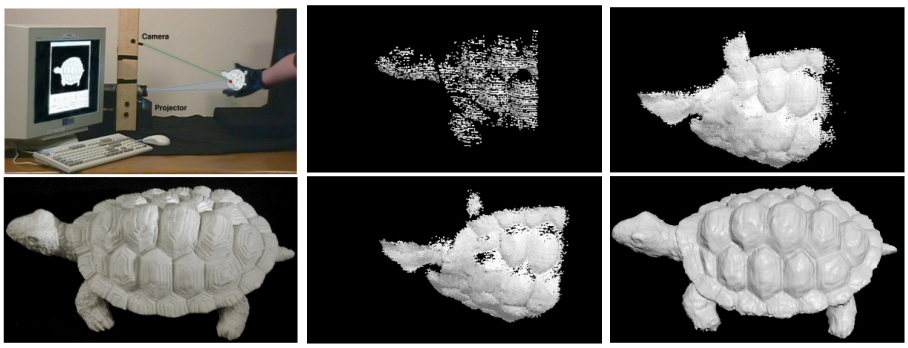


Figure 2.8: In-hand reconstruction from [Rusinkiewicz and Levoy, 2001]. The left column shows the general setup and the object of interest while the other images depict the gradual fusion of consecutive views. The black gloves allow for easy background subtraction.

color filtering) or 3D (e.g. plane fitting) we can determine which pixels depict the object of interest and run simple foreground extraction. After determining 3D point geometry (either by triangulating 2D point correspondences between views or by explicitly providing reprojected depth data) the information is fused into a common frame. In Figure 2.7 the fusion is then merely a sparse collection of partial colored 3D points, transformed into the same reference frame.

Although the dynamic approach (see Figure 2.8 for an in-hand scanning example) is naturally more appealing, proper background segmentation as well as transformation estimation is hard to accomplish. To remove the background, the works [Rusinkiewicz et al., 2002, Weise et al., 2011] use colored gloves which are detected and filtered out. The extrinsic transformation between two scan frames is then obtained by running the ICP method, followed by a global refinement step. Such setups work well if the object has rich geometric structure and the inter-frame movement is small. Methodologically, it fails for objects with poor geometrical and textural discriminance (or even symmetries) since the registration between frames becomes unreliable. The work [Krainin et al., 2011] follows the same idea, but uses a robot and its arm pose to recover the transformation between frames without visual estimation. When scanning larger objects [Zollhöfer et al., 2014, Newcombe et al., 2015, Slavcheva et al., 2018], the background segmentation is done via color filtering and/or depth thresholding.

From the examples above it should be evident that every single mentioned aspect is important for correct reconstruction. While it is usually straightforward to calibrate the intrinsics of a camera, it is much harder to determine the extrinsic relation of viewpoints (especially when these relations cannot be estimated from artificial or natural fiducials). In many scenarios these relations must therefore be robustly estimated from motion observed between consecutive frames, either by matching 2D image keypoints in 3D [Endres et al., 2012] or densely by ICP variants [Besl and McKay, 1992, Rusinkiewicz and Levoy, 2001]. An early work on automatic registration of multiple 3D scans can be found in [Huber and Hebert, 2003], where range scans of an object are transformed into

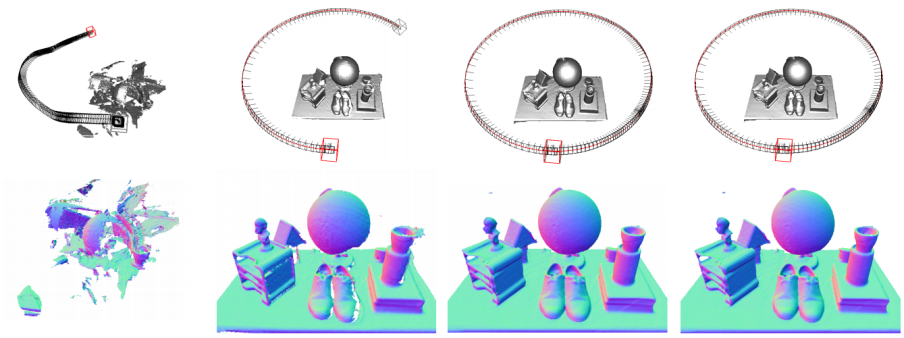


Figure 2.9: Incremental KinectFusion reconstruction [Newcombe et al., 2011]. From left to right more scans get fused into one global signed distance field representation while smoothing out noise.

partial meshes, matched pairwise and put into a global graph optimization problem to find the most consistent connected subgraph. In [Makadia et al., 2006], the authors solve for the alignment by defining correlation functions and computing Fourier transforms with a subsequent verification stage. These works register data which is represented sparsely of either points or surface approximations.

Opposed to sparse point representations, one can also represent depth via 3D signed distance fields [Curless and Levoy, 1996], which have the advantage of being continuous function with smoothing properties. They have been subsequently used in many recent works including scene reconstruction and camera tracking [Graber et al., 2011, Newcombe et al., 2011, Whelan et al., 2013, Bylow et al., 2013, Ren et al., 2013]. The dense nature of distance fields allows to introduce operators working on functions while still being able to extract a (possibly sparse) surface as a level-set. Furthermore, it has also been shown in [Rouhani and Sappa, 2013] that a richer data representation can, in fact, help in registering when moving from simple point-based metrics to ones using implicit shape representations.

KinectFusion and its variants [Newcombe et al., 2011, Whelan et al., 2013, Bylow et al., 2013] estimate the motion between frames via ICP-based energies but fuse the data into one continuous dense distance field. This produces static scene reconstructions of high fidelity (see Figure 2.9). Due to the inability of ICP to cope with symmetric or non-distinctive geometry, KinectFusion often fails in these scenarios. Furthermore, it still requires the user to specify volume boundaries and to sometimes deal with post-processing steps after meshing when singling out objects. As an alternative to ICP, visual odometry approaches use both color and depth to estimate the warp that encodes the camera motion [Steinbrucker et al., 2011, Kerl et al., 2013]. They can overcome situations where ICP fails and thus, provide for better estimations of camera movement. However, many reconstruction approaches often assume a static scene which prohibits the displacement of objects during scanning. As a result, incomplete object geometry is usually obtained with the bottom or some self-occluded parts

missing from the reconstruction. In chapter 3 we will present a method that will combine KinectFusion ideas with visual odometry to create coherent, full reconstructions of rigid 3D objects.

For non-rigid object reconstruction, the works [Zollhöfer et al., 2014] and [Slavcheva et al., 2018] start from a initial scan template and track the deformation either sparsely or variationally while DynamicFusion [Newcombe et al., 2015] as well as [Innmann et al., 2016] avoid templates and keep track of connected correspondences instead.

As mentioned, signed distance fields can be regarded as a special variant of level-set methods, surveyed in [Jones et al., 2006], and apart from object reconstruction, have been used in many other fields, e.g. fluid dynamics [Bargteil et al., 2006, Losasso et al., 2004, Losasso et al., 2006] where the physical properties of the model act upon the level-set function via PDEs. As mentioned before, in contrast to explicit representations which can entail topological difficulties as well as rendering mathematical operations harder to implement, level-sets can implicitly represent arbitrary shapes and are therefore often preferred in these domains. Nonetheless, optimizing in volumetric data always is costly and related work tackle it in different ways.

[Zach et al., 2007] conducts variational data fusion and uses a run-length encoding that allows for fast decompression of the input data but does not address the problem of the structurally-changing minimizer during optimization. In a follow-up work [Zach et al., 2008], the authors propose a coarser quantization of the SDF values and introduce a point-wise histogram based problem. In [Schroers et al., 2012] the authors suggest to modify the data term such that one tries to be similar to the point-wise median. The authors of [Popinet, 2003] claim to have the first single-pass hierarchical-based approach for incompressible Euler equations based on multi-grids, although earlier works (e.g. [Strain, 1999]) already focused on tracking moving interfaces with tree-based structures. In [Losasso et al., 2004, Losasso et al., 2006], the authors deal with spatially adaptive techniques for incompressible flow and state their surprise about the high accuracy of Octree-based mesh refinement even for small-scale structures. The works [Steinbrucker et al., 2013, Chen et al., 2013, Zeng et al., 2013] use dynamic Octree-based representations to store scene geometry but do no conduct any elaborate schemes for the integration since their main interest is mapping and efficient storage/updates for large-scale problems. [Houston et al., 2006] introduces a data structure that includes a hierarchical partitioning where each cube uses a run-length encoding. Although very efficient in storage and lookup, online restructuring is rather slow and therefore less suited for iterative structural changes. In [Nießner et al., 2013] and later [Klingensmith et al., 2015], the authors present an alternative to hierarchical partitioning by hashing the SDF geometry to enable near-constant time lookups. All of the above mentioned works are focusing only on efficient storage and processing while paying little attention to quantitative reconstruction quality. In chapter 4 we will present our extension for fusing range data such that we perform efficiently in terms of memory and run-time while keeping a high reconstruction accuracy.

We will briefly explain the context of the following two chapters before giving in-more detailed explanations in their respective sections.

Reconstruction with Colored Signed Distance Fields In the second chapter, based on the work *Coloured signed distance fields for full 3D object reconstruction* (BMVC2014), we present a full 3D reconstruction pipeline combining visual odometry and KinectFusion ideas. We propose a novel 3D object reconstruction framework that is able to fully capture the accurate coloured geometry of an object using an RGB-D sensor. Building on visual odometry for trajectory estimation, we perform pose graph optimisation on collected keyframes and reconstruct the scan variationally via coloured signed distance fields. To capture the full geometry we conduct multiple scans while changing the object's pose. After collecting all coloured fields we perform an automated dense registration over all collected scans to create one coherent model. We show on eight reconstructed real-life objects that the proposed pipeline outperforms the state-of-the-art in visual quality as well as geometrical fidelity.

Efficient Variational Depth Data Fusion with Octrees The third chapter is based on *An Octree-Based Approach towards Efficient Variational Range Data Fusion* (BMVC2016). Volume-based reconstruction is usually expensive both in terms of memory consumption and runtime. Especially for sparse geometric structures, volumetric representations produce a huge computational overhead. In this chapter we will build on our previous work and present an efficient way to fuse range data via a variational Octree-based minimization approach. We transform the data into Octree-based truncated signed distance fields and show how the optimization can be conducted on the newly created structures. The main challenge is to uphold speed and a low memory footprint without sacrificing the solutions' accuracy during optimization. We explain how to dynamically adjust the optimizer's geometric structure via joining/splitting of Octree nodes and how to define the operators. We evaluate on various datasets and outline the suitability in terms of performance and geometric accuracy.

Chapter 3

Reconstruction with Colored Signed Distance Fields

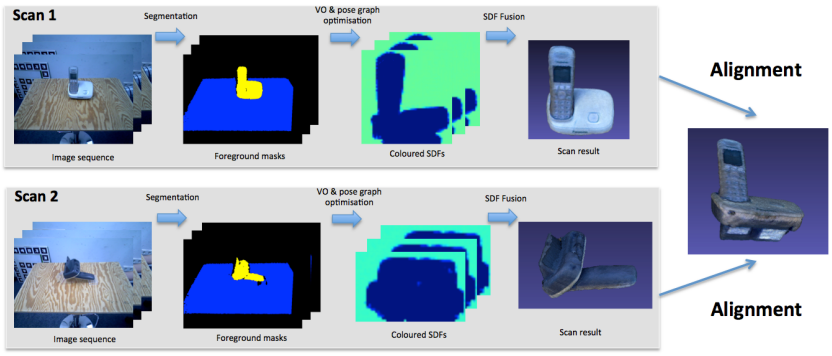


Figure 3.1: Each scan sequence is masked, pose-optimized and fused to create a model. Then all scans get aligned to one coherent model.

We propose a full reconstruction framework with a RGB-D sensor, requiring no marker boards and allowing for objects to be displaced during scanning. It consists of multiple stages which are outlined in the next paragraphs. Even though parts of the method are existing in the related works, the proposed framework is unique and provides a novel fusion and registration procedure for CSDFs resulting in complete 3D models with high precision. We will also demonstrate that our method is applicable to a large variety of objects.

Firstly in our pipeline, we estimate the camera trajectory via RGB-D visual odometry while moving the support surface, collecting keyframes along the way and globally refining the trajectory with a pose-graph optimization after detecting loop closure. After one full scan and pose refinement, we refer to our final result as a hemisphere $\mathcal{H}$ consisting of a number of RGB-D keyframes with asso-

ciated poses. We create a 3D model ϕ by fusing the data in a variational fashion using colored signed distance fields and an approximate L^1 minimization. Usually, one such scan does not expose the full geometry of the object. To this end, we propose to create multiple scans of the same object but placed differently in order to reveal hitherto unseen parts, thus acquiring multiple hemispheres $\mathcal{H}_j$. Then the transformations Ξ_j that map the models from all hemispheres to the first one $\mathcal{H}_0$ needs to be determined. In order to retrieve those Ξ_j, we use the reconstructed models ϕ_j and align them automatically using a dense approximate-L^1 registration framework.

3.1 3D object scanning

We follow the concept of [Dimashova et al., 2013] which is now shortly presented and visualized in Figure 3.1. We create a sequence with a fixed sensor and a rotating support surface. In order to reliably assess the camera movement, a separation of foreground from background has to be performed. We define a RGB-D sensor pair $[I : \Omega_2 \to [0,1]^3, D : \Omega_2 \to \mathbb{R}^+]$ and the camera projection $\pi : \mathbb{R}^3 \to \Omega_2$. By fitting a plane into the cloud data $C := \pi_D^{-1}$ and computing the prism spanned by the support plane along its normal direction, the 3D points lying inside the prism are determined and projected into the image plane to create foreground segmentation masks.

Then we estimate the camera transformation via visual odometry using RGB-D data [Steinbrucker et al., 2011]. The goal is to compute the rigid-body movement $\Xi \in SE(3)$ of the camera between two consecutive sensor pairs $[I_0, D_0]$,$[I_1, D_1]$ by maximizing the photo-consistency

$$E(\Xi) = \int_{\Omega_2} [I_1(w_\Xi(\mathbf{x})) - I_0(\mathbf{x})]^2 dx \tag{3.1}$$

with a warp function $w_\Xi : \Omega_2 \to \Omega_2$, defined as $w_\Xi(\mathbf{x}) = \pi_{D_0}(\Xi \cdot \pi_{D_1}^{-1}(\mathbf{x}))$ that transforms and projects the colored point cloud from one frame into the other. To solve this least-squares problem, the authors employ a Gauss-Newton approach with a coarse-to-fine scheme. We refer to [Steinbrucker et al., 2011] for details.

The advantage of using RGB-D visual odometry as opposed to plain ICP (as for example done in [Newcombe et al., 2011]) is that we can handle scenes and/or objects which suffer from poor geometrical discriminance. Although the odometry approach is naturally also prone to drifting, it showed to be far more reliable for the problem at hand as long as the support surface and the object exhibit a fair amount of texture.

While estimating the frame-wise transformation update, we store keyframe pairs $[I_i, D_i]$ plus pose P_i after having seen a sufficient amount of change between pose P_i and pose P_{i-1} from the last stored keyframe (in our implementation, 10 degrees and 10cm). To detect loop closure, we make sure that we have already observed a substantial amount of transformation in comparison to the initial frame (330 degrees or 2m) and then start comparing incoming pairs with the initial pair via color ICP and the inlier ratio. Eventually, we run a pose-graph optimization using the g2o framework [Kummerle et al., 2011] in order to

refine the camera poses while we base the error measure on the visual odometry energy.

3.2 Variational sensor data fusion

Given one hemisphere $\mathcal{H} = \{(I_i, D_i, P_i)_i\}$ consisting of sensor pairs and poses, we fuse the data into a coherent model. Analogously to [Zach et al., 2007, Kubacki et al., 2012, Schroers et al., 2012, Ren and Reid, 2012], we cast data into volumetric fields $f_i : \Omega_3 \subset \mathbb{R}^3 \rightarrow \mathbb{R}$ in order to smoothly integrate them into one fused model [Curless and Levoy, 1996]. Note that we will write partial fields with f and final fields with ϕ to stay consistent with our defined notation. The creation of the truncated SDFs follows the description from Section 2.2.1. and we have found that an uncertainty parameter of $\delta = 2mm$ works well for our application and fixed it throughout this work. Since projective SDFs are dense, they wrongly encode obstructed spatial area. In order to signify what parts of the SDF we trust, we endow every f_i with a binary weighting function $w_i : \Omega_3 \rightarrow \{0, 1\}$ that selects volumetric parts for the fusion process:

$$w_i(\mathbf{X}) = \begin{cases} 1 & \text{if } D_i(\pi(\mathbf{X})) - ||\mathbf{X} - P_i^{xyz}|| < -\eta \\ 0 & \text{else} \end{cases} . \tag{3.2}$$

The parameter η defines how much has been seen behind the observed surface and assumed to be solid (we fixed $\eta = 1cm$). Since we are interested in recovering the color of the object as well, we furthermore define a color volume $c_i : \Omega_3 \rightarrow [0, 1]^3$ as:

$$c_i(\mathbf{X}) = I_i(\pi(\mathbf{X})). \tag{3.3}$$

We will refer to the joint representation $[f_i, c_i]$ as CSDF. The goal now is to recover functions u, v which hold the object's reconstructed geometry and coloring, respectively. Following [Schroers et al., 2012], we cast the problem into a variational energy optimization formulation where we seek the minimizers of the functional

$$\mathcal{E}(u, v) = \int_{\Omega_3} \left[\mathcal{D}(\mathbf{f}, \mathbf{w}, \mathbf{c}, u, v) + \alpha\mathcal{S}(\nabla u) + \beta\mathcal{S}(\nabla v)\right] \, d\mathbf{x} \tag{3.4}$$

with a data term $\mathcal{D}$ that strives to uphold the solution's fidelity to all the observations $\mathbf{f} = \{f_1, ..., f_n\}, \mathbf{c} = \{c_1, ..., c_n\}$ and two regularizers $\mathcal{S}(\nabla u)$ and $\mathcal{S}(\nabla v)$, weighted with α and β, respectively, that force the minimizers to be smooth. Note that, in contrast to the original work [Schroers et al., 2012], which only fuse the geometrical fields, we also include color information into the formulation and solve simultaneously for both.

A suitable data term for many problems in reconstruction and segmentation usually involves an outlier-robust L^1-norm whereas for regularization purposes the total variation (TV) of the function is often employed:

$$\mathcal{D}(\mathbf{f}, \mathbf{w}, \mathbf{c}, u, v) = \frac{1}{\epsilon + \sum_i w_i} \sum_i w_i \cdot (|u - f_i| + |v - c_i|) \ , \ \mathcal{S}(\nabla u) = |\nabla u| \tag{3.5}$$

Due to the problematic aspect of solving such energies, specific minimization schemes are employed (e.g. a ROF-variant [Zach et al., 2007] or (iterated) primal-dual solutions [Graber et al., 2011, Ochs et al., 2013]). An alternative has been proposed in [Schroers et al., 2012], where the problematic terms have been replaced with a smooth approximation $\Gamma(x) := \sqrt{x^2 + \epsilon}$ ([Lee et al., 2006]). We define it similarly:

$$\mathcal{D}(\mathbf{f}, \mathbf{w}, \mathbf{c}, u, v) = \Gamma(\sum_i w_i)^{-1} \sum_i w_i \cdot (\Gamma(u - f_i) + \Gamma(v - c_i)) \ , \ \mathcal{S}(\nabla u) = \Gamma(|\nabla u|) \tag{3.6}$$

where we regard the weighted approximate absolute differences together with an additional normalization factor and an approximate TV-regularizer. This regularizer penalizes the perimeter of the level sets and therefore leads to the removal of isolated small-scale features and the shaping of a low-genus isosurface of u.

With this strictly convex and differentiable formulation, the global minimizers can be found at the steady state of the gradient descent equations

$$\partial u = \alpha \cdot \operatorname{div}(\mathcal{S}'(\nabla u)) - \frac{\partial \mathcal{D}}{\partial u}(\mathbf{f}, \mathbf{w}, \mathbf{c}, u, v) \tag{3.7}$$

$$\partial v = \beta \cdot \operatorname{div}(\mathcal{S}'(\nabla v)) - \frac{\partial \mathcal{D}}{\partial v}(\mathbf{f}, \mathbf{w}, \mathbf{c}, u, v) \tag{3.8}$$

which we determine and denote as u^*, v^*. We now also define our reconstructed CSDF $\phi : \Omega_3 \to [-1, +1] \times [0, 1]^3$ with $\phi = [u^*, v^*]$. This presented fusion method is applied to all collected hemispheres $\mathcal{H}_j$ to produce a corresponding model, i.e. CSDF ϕ_j.

3.3 Automatic colored SDF alignment

Let us resume to the problem of aligning all the models ϕ_j from hemispheres $\mathcal{H}_j$, which we reduce to solving pairwise problems of aligning models ϕ_j to ϕ_0. Thus, a 3D rigid-body transformation Ξ_j needs to be determined. We are faced with six dimensions of freedom, and as outlined in the background chapter, choose a minimal representation of our transformation. This means we parametrize via a twist vector $\xi = [\omega_x, \omega_y, \omega_z, t_x, t_y, t_z]^T \in \mathbb{R}^6$, recovering the transformation $\Xi = \exp(\hat{\xi})$ with the goal that, for every point $\mathbf{X}$, we achieve $\phi_0(\mathbf{X}) = \phi(\Xi_j(\mathbf{X}))$. Here we write $\Xi(\mathbf{X}) : \mathbb{R}^3 \to \mathbb{R}^3$ as a short-hand form for the homogeneous transformation parametrized by ξ and applied to point $\mathbf{X}$. Since exact solutions usually do not exist in practice, we try to minimize their distance instead by an energy formulation with an appropriate measure M:

$$\mathcal{E}(\Xi) = \int_{\Omega_3} M(\phi_0(\mathbf{X}), \phi_j(\Xi(\mathbf{X}))) \ d\mathbf{X}. \tag{3.9}$$

The striking difference to KinectFusion-based approaches is that we tackle the registration problem in a fully continuous global manner while solving an algebraic error, inspired by [Paragios et al., 2002, Rouhani and Sappa, 2013]. It is noteworthy that, in the works [Kubacki et al., 2012, Bylow et al., 2013], the authors go half the way by formulating a point-SDF distance measure. Our

approach can be regarded as an algebraic (i.e. non-geometric) generalization of ICP to dense volumetric representations. One differentiable measure that one can use here is the L^2-norm:

$$\mathcal{E}(\Xi)_{L^2} = \int_{\Omega_3} \frac{1}{2}(\phi_0(\mathbf{X}) - \phi_j(\Xi(\mathbf{X})))^2 \; d\mathbf{X}. \tag{3.10}$$

In order to optimize the given non-linear energy, the mentioned related works (using an SSD measure over points) usually rely on a Gauss-Newton scheme, where they iteratively linearize the problem at a certain point and solve linear equation systems. Since tracking speed in real-time is an important issue for them, their method is the most appealing due to its convergence speed and small incremental camera changes. We, on the other hand, are interested in precise alignment with larger transformations and therefore propose a more robust energy that employs the approximate L^1-counterpart:

$$\mathcal{E}(\Xi)_{L^1} = \int_{\Omega_3} \Gamma(\phi_0(\mathbf{X}) - \phi_j(\Xi(\mathbf{X}))) \; d\mathbf{X}. \tag{3.11}$$

In theory, one could use many different differentiable distance measures like correlation ratios or mutual information. We apply a gradient descent scheme to update the transformation parameters. Starting with an initial transformation Ξ^0, we iteratively solve (depending on the measure):

$$\nabla\xi = \quad -\tfrac{1}{|\Omega_3|}(-\tfrac{\partial\phi_j}{\partial\Xi^i} \cdot \tfrac{\partial\Xi^i}{\partial\xi}) \cdot (\phi_0 - \phi_j(\Xi^i)) \quad (L^2) \tag{3.12}$$

$$\nabla\xi = \quad -\tfrac{1}{|\Omega_3|}(-\tfrac{\partial\phi_j}{\partial\Xi^i} \cdot \tfrac{\partial\Xi^i}{\partial\xi}) \cdot \Gamma'(\phi_0 - \phi_j(\Xi^i)) \quad (L^1) \tag{3.13}$$

$$\Xi^{i+1} = \quad \exp(\tau \cdot \nabla\hat{\xi}) \cdot \Xi^i \tag{3.14}$$

having a twist update $\nabla\xi$ for either energy with the Jacobian $\frac{\partial\phi_j}{\partial\Xi^i} \cdot \frac{\partial\Xi^i}{\partial\xi}$, a gradient step size τ and an additional normalization $\frac{1}{|\Omega_3|}$ to ensure a proper numerical update.

The proposed energy admits local optima and therefore depends on the initialization While we generally observe a robust convergence, we still have the problem of large rotational differences in-between hemispheres. To this end, we search in a spherical grid by sampling spherical coordinates in the ranges $\theta \in (0, \pi), \kappa \in (0, 2\pi)$ in discrete steps and run the alignment until convergence. To speed up the registration process, we employ a coarse-to-fine pyramid scheme over three levels where down-sampled models are roughly aligned and the resulting alignment is taken as the initialization into the next pyramid stage.

To assess the feasibility of a given solution, we raytrace both SDFs at their 0-level-set from densely-sampled views and do a pixel-wise comparison of the color and depth renderings. Since our colored SDFs can contain spurious noise inside and outside the object and since we introduced a discretization error due to voxel grids, simply evaluating the energy can be misleading. The raytrace comparison showed to be more reliable since it neglects values which are not on the object's surface and it does not suffer from the discretization in the same amount.

After deciding for the best alignment Ξ_j, we fuse both hemispheres by simply merging their elements into the first $\mathcal{H}_0 := \{(I_i^0, D_i^0, P_i^0)_i, (I_k^j, D_k^j, \Xi_j \cdot P_k^j)_k\}$

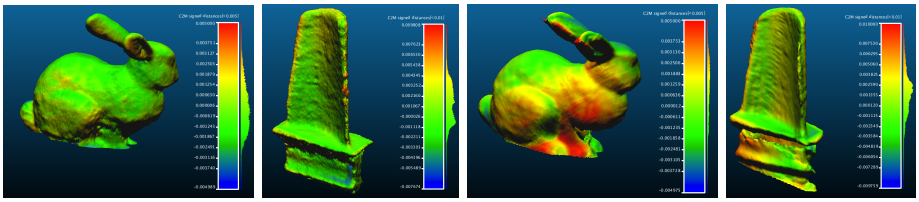


Figure 3.2: Reconstruction accuracy in meters in respect to ground truth data of *bunny* and *turbine*. Left: our approach. Right: KinectFusion. Our method is able to recover finer details due to optimizing both the pose graph and the sensor data integration.

while transforming the poses. Then we start again by reconstructing the object from $\mathcal{H}_0$, having now more views, and align it with the model ϕ_{j+1} of the next hemisphere $\mathcal{H}_{j+1}$. Eventually, we run the Marching Cubes algorithm [Lorensen and Cline, 1987] to extract a mesh at the 0-level set of ϕ_0.

3.4 Evaluation

The proposed algorithm has been used to reconstruct 8 real-life objects, namely *book*, a 3D print of Stanford's *bunny*, *drill*, *mango*, *milk*, *phone*, *tape* and *turbine*. They were placed on a table and two sequences of around 800 images have been recorded for each object.

3.4.1 KinectFusion versus our method

We reconstructed by using our method and the RecFusion software being a commercial KinectFusion variant. The voxel size was always fixed to $1mm$ due to RecFusion's constrained scan settings and we ran the methods on exactly the same sequences. The reconstruction results are presented in Figure 3.3. Even though KinectFusion performed well, it failed for the objects *mango*,*milk* and *tape* due to poor geometry leading to a failure in camera tracking. We were able to reconstruct every object with full geometry and rich texture. For the two models *bunny* and *turbine* ground-truth data was available and was used to measure the geometrical error of the reconstructions (see Figure 3.2). To be fair for the latter, we compared the KinectFusion results with ours only by reconstructing from one hemisphere. Thus, both methods worked on the same data since we wanted to demonstrate the accuracy obtained with our optimization pipeline. We clearly boost the geometrical fidelity due to the pose graph optimization and the L^1 sensor fusion.

3.4.2 L^1 versus L^2 registration

In order to show the superiority of the approximate L^1 registration in comparison to L^2, we ran the alignment (given an initial transformation) on all reconstructed objects. The rotations were sampled from spherical coordinates

Figure 3.3: Reconstructions of the eight objects. Each pair depicts the results from KinectFusion on the left and our approach on the right. We clearly recover richer texture as well as geometry. We are even able to fully reconstruct symmetric objects like the tape object or geometrically poor objects like the mango. Also note the textural fidelity of mostly every object that was reconstructed with our method.

Object	book	bunny	drill	mango	milk	phone	tape	turbine
# Pos. L^1 runs	4	6	7	2	3	6	11	3
# Pos. L^2 runs	4	5	6	2	3	5	10	3
Avg. L^1 iters	289	204	363	102	282	261	120	425
Avg. L^2 iters	344	284	501	134	352	462	148	411

Figure 3.4: Registration results using both measures for the given objects. A total of 16 runs with different rotational initializations have been performed. The L^1 registration outperforms the L^2 formulation both in terms of its wider convergence basin and speed.

Figure 3.5: Registration for *phone* and *bunny* visualized as a difference image of ray-casted volumes. From left to right: Target, initialization, results for L^1 and L^2. The L^1-energy converged in these cases.

in discrete steps (resulting in 16 runs) and the SDFs were mean-centered. We measured both the number of successful runs (i.e. global convergence) and the number of average gradient descent iterations over all successful runs. Figure 3.4 summarizes the registration results. We can observe that the L^1 formulation consistently leads to an energy that allows for easier global convergence while reducing the number of gradient descent steps. Figure 3.5 shows a typical case where the L^1-energy was able to globally converge whereas the L^2-energy got stuck locally.

3.5 Conclusion

We presented a novel pipeline for full coloured 3D reconstruction. We showed that by using optimization in all stages, namely: camera pose graph, data fusion and registration, we can reach very high fidelity for both texture and geometry. Furthermore, our method is able to deal with a larger variety of objects since our camera tracking, relying on textured tables, overcomes typical pitfalls of geometry-based ambiguities.

Chapter 4

Efficient Variational Depth Data Fusion with Octrees

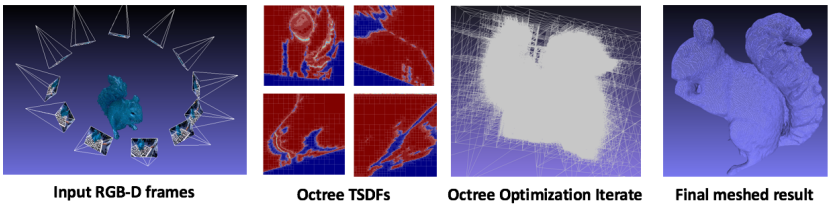


Figure 4.1: Our work deals with robust variational fusion of range scans. Given a sequence of input frames, we optimize over Octrees that represent transformed input TSDFs into a common, constantly evolving Octree to finally retrieve a geometrically accurate and smoothed meshed reconstruction.

Hierarchical space partitioning schemes, like kD-trees [Bentley, 1975] or Octrees [Meagher, 1982], can tremendously increase query performance for static geometries but need to be properly updated for dynamic changes occurring inside the volume. Obviously, employing partitioning schemes during an optimization must be carefully designed to avoid accumulation of quantization errors which lead to inaccurate solutions. In this work, we build our optimization around Octrees which recursively divide up the space into eight equally-sized cubes according to split and join rules. Octrees are established in many fields and are often used to alleviate computational burdens. Recent work ([Steinbrucker et al., 2013, Chen et al., 2013, Zeng et al., 2013]) uses Octrees for range data integration to map the environment, but employs simple update rules to encompass newly seen data without any optimization whatsoever. For simulation problems these partitioning structures are usually of static auxiliary nature (e.g. accelerating point/surface look-ups, [Calakli and Taubin, 2011, Losasso et al., 2004]) and get discarded or recomputed after each iteration.

Our novel contribution is to use an Octree as the main optimization structure instead and we present a viable way to robustly fuse TSDFs in a variational approach, similar to [Zach et al., 2007, Kehl et al., 2014], while dynami-

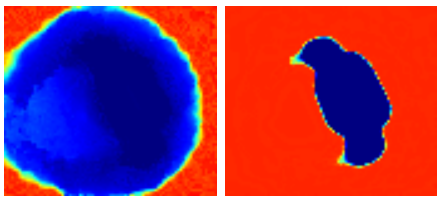


Figure 4.2: Slicing through a dense TSDF with a large $\delta = 20$ cm (left) and a very tight $\delta = 2$ mm (right). The narrow band at the real object surface is clearly visible in the right image. We want to numerically focus on this interface while neglecting the uniform areas.

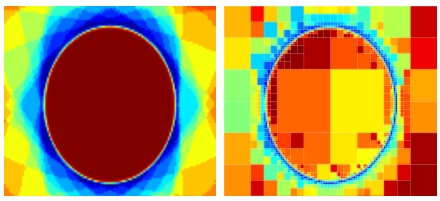


Figure 4.3: Left: Slicing through a dense TSDF (left) and its Octree-version (right). Blue areas close to the surface possess a finer Octree-resolution since these represent the narrow band during optimization and should therefore be similar to the real TSDF values.

cally adapting the Octree's iterative structure to be faster and memory-efficient. We address the actual creation of the Octrees given initial range maps, the proper definition and calculation of mathematical quantities as well as correct numerical updates that include the iterative reorganization of the solution's hierarchical partitioning after each step.

Firstly, we discuss the influence of the scaling factor on the computation of TSDFs as well as the transformation into their Octree-representations. Secondly, we introduce the new problem formulation and give a way to efficiently solve it via node-wise split/join rules and a fast traversal technique.

Note that in contrast to to the previous chapter which fuses into RGB-D volumes, we solely focus here on the geometric minimizer because the space partitioning is not applicable to color volumes. Instead, we compute the coloring after meshing of the 0-isosurface similarly to [Whelan et al., 2013].

4.1 TSDF-Octree generation

Let us remember that the creation of truncated SDFs from Section 2.2.1 included a decision function

$$\psi(d) = \begin{cases} sgn(d) & \text{if } |d| > \delta \\ \frac{d}{\delta} & \text{else} \end{cases} \tag{4.1}$$

that would map a computed distance d to a truncated signed value $\psi(d) \in \{-1, +1\}$. The supplied scaling factor δ serves as an uncertainty band and should be well-chosen both to compensate for measurement noise and to clearly divide between outer and interior space (see Figure 4.2). Strictly speaking, areas which are far apart from the interface consume memory and run-time during optimization without having a drastic influence on the optimizer's object surface. Our goal is to ensure that these spaces remain computationally inexpensive at all times during the optimization without impairing the final solution. To this end, we transform our problem to work with space-partitioned entities which

can adapt to the changing narrow band of our iterated solutions.

Octree construction We construct TSDF-Octrees f_i^* from f_i in a top-to-bottom manner. Starting from root node n, we define the spread s of values subsumed by node n in f as

$$s_f(n) = \left| \max_{\mathbf{X} \in \Omega_3(n)} f(\mathbf{X}) - \min_{\mathbf{X} \in \Omega_3(n)} f(\mathbf{X}) \right| \tag{4.2}$$

with $\Omega_3(n)$ being the subvolume that node n represents. Initially, $\Omega_3(n) = \Omega_3$ and the spread will be maximal. From here we recursively apply a splitting rule: if the spread $s_f(n)$ at node n is higher than a threshold $\tau = 0.1$, we subdivide n into eight children and proceed further down. This is recursively repeated as long as the condition is fulfilled or until we reach a maximum Octree depth $D_{\max}$, corresponding to a pre-defined minimum metric voxel size. After the partitioning we propagate the means upwards from the leafs to all inner nodes to speed up computations during later optimization. See Figure 4.3 for a visual comparison.

4.2 Octree-based variational optimization

Similar to the last chapter we fuse all range maps into one volume by finding the minimizer of

$$\mathcal{E}(u) := \int_{\Omega_3} D(\mathbf{f}, \mathbf{w}, u) + \lambda S(\nabla u) \, d\mathbf{x} \tag{4.3}$$

where we weight data fidelity against a regularization with a smoothness parameter λ. [Zach et al., 2007] employs an outlier-robust L^1 data term $D(\mathbf{f}, \mathbf{w}, u) := \sum_i w_i \cdot |u - f_i|$ together with a Total Variation (TV) regularizer $S(\nabla u) := |\nabla u|$ which has the advantage of penalizing the perimeter of the level sets in u and in combination, TV-L^1 induces a pure geometric regularization, as found in [Chan et al., 2006]. Unfortunately, the non-differentiability requires elaborate solving schemes and we thus follow [Schroers et al., 2012, Kehl et al., 2014] by tightly approximating both terms with differentiable quantities

$$D(\mathbf{f}, \mathbf{w}, u) := \sum_i w_i \cdot \Gamma((u - f_i)^2) \quad , \quad S(\nabla u) := \Gamma(|\nabla u|^2) \tag{4.4}$$

with $\Gamma(x^2) := \sqrt{x^2 + \epsilon^2}$ as introduced in the last chapter.

An important aspect is the data term normalization since in its current form the functional puts more emphasis on the data term if the number of range images increases. Instead of dividing by the number of images we divide pointwise by the accumulated weight to achieve a spatially consistent smoothing, regardless of how often a voxel has been seen [Schroers et al., 2012]. Altogether, this strictly-convex functional can now be solved by gradient descent and in our Octree-based variant, we furthermore replace all quantities with their space-partitioned counterparts to finally retrieve

$$\mathcal{E}(u^*) := \int_{\Omega_3} \frac{D(\mathbf{f}^*, \mathbf{w}^*, u^*)}{\sum_i w_i^* + \gamma} + \lambda S(\nabla u^*) \, d\mathbf{X}, \tag{4.5}$$

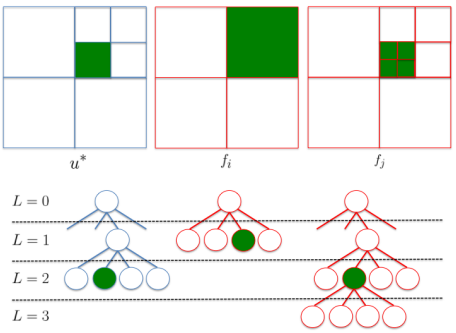


Figure 4.4: Data term computation. Standing at the green node n in u^* at level 2, we query all TSDFs at the same spatial location. Either the level is not available (f_i) in which case we fetch the node that spatially subsumes n or the level is the same/deeper (f_j) and we fetch the pre-computed value at the same level.

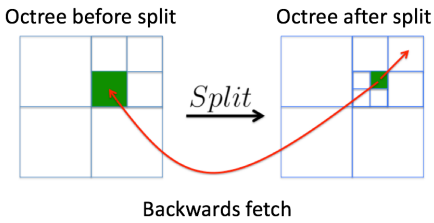


Figure 4.5: Regularizer computation. The red arrows symbolize which nodes are needed to get the divergence for the green node on the right side. Fetching forward differences is always possible during the recursive run. For backward differences we avoid numerical errors by storing the node value before its split.

and instead solve for u^* with a small γ in the normalizer to avoid division problems for unseen voxels. To optimize Equation 4.5, we determine the steady state of our PDE

$$\frac{\partial \mathcal{E}}{u^*} = \lambda \operatorname{div}(S_{\nabla u^*}(\nabla u^*)) - \frac{D_{u^*}(\mathbf{f}^*, \mathbf{w}^*, u^*)}{\sum_i w_i^* + \gamma}. \tag{4.6}$$

Note that we, strictly speaking, optimize a new u^* in each iteration since we constantly change the structure of our iterate. Nonetheless, this showed to be not a problem in practice since we observed a proper convergence in every case. We will now focus on clarifying how we evaluate above terms and how to conduct the actual optimization in the Octree.

Optimization on the Octree We conduct the optimization by having at all times only one version of u^* in memory and adjusting the structure while we recursively traverse into each node of u^*. This means that instead of integrating point-wise over the volume, we start from the root and run along the tree while conducting our computations/restructuring on it before proceeding to the next node in the volume in the same pass. To mathematically facilitate the notation, we allow our TSDFs to take nodes as arguments.

For the repartitioning during optimization, we retrieve the gradient descent update Δu_t^* at iteration t for u_t^*. Since the update might encompass larger numerical changes, we need to reflect this by restructuring u_{t+1}^* before applying the update such that no crucial information is lost. In contrast to [Losasso et al., 2004, Losasso et al., 2006] we do not refine the cube resolution of the Octree simply based on their spatial distance to the narrow band but

rather refine them based on numerical values. For an Octree node n during our run, we compute, together with a gradient step size ξ, the new value $\Delta n = u_t^*(n) + \xi \cdot \Delta u_t^*(n)$ and decide for the repartitioning together with a splitting threshold τ_s, a joining threshold τ_j and two rules:

- if n is a leaf of the Octree and $|\Delta n| < \tau_s$, we split and recurse into the children
- else n is not a leaf of the Octree and we check if $|\Delta n| > \tau_j$. If this holds, we recursively conduct the same check for the children and if successful, we join these children only if furthermore they all hold values of equal sign. Otherwise, an implicit surface passes through these nodes and joining them might rupture it.

In order to compute node-wise expressions in our Octree-TSDFs, we simply fetch the corresponding values from the tree nodes that represent the subvolume at this position: if $u^*(n)$ resides at tree level L, we either fetch the pre-computed corresponding values $f_i(n)$ and $w_i(n)$ at level L, if the level exists, or take the closest leaf n' that spatially subsumes n (see Figure 4.4). The f_i can be efficiently queried while moving alongside n in each Octree.

For the more complex quantity, the gradient ∇u^*, we use forward differences which need to be fetched in a neighborhood around each node n. This can be easily accomplished during the same pass since we can spatially look-up all nodes ahead of n which have not been touched yet. To compute the divergence, we also need to be able to compute backward differences. In our approach we want to be fast and therefore want to accomplish one optimizer iteration in one pass through the Octree. Thus, we immediately restructure all visited nodes and would therefore induce numerical errors if we fetch backwards during the same pass. To remedy that we store for a node its value before splitting such that the computation is proper (see Figure 4.5). Note that this is not applicable when joining a node since it would need to carry a history of all its children values. However, due to our splitting rules, joins never happen at interfaces and we thus can discard this otherwise problematic issue.

Meshing and coloring As a final step, we again apply Marching Cubes to extract the 0-level isosurface to retrieve a mesh and compute the coloring similarly to [Whelan et al., 2013]. Furthermore, to supply color to unseen parts, we iteratively propagate the colors along triangle boundaries and blend the final color in respect to the neighboring, colored vertices and their dot products.

4.3 Evaluation

The method was implemented in C++ and the experiments were conducted on a CPU with 32GB of RAM. We ran our experiments on different kinds of data: firstly, we synthetically rendered 31 views of a sphere to measure our loss in accuracy on perfect, noise-free data. Secondly, we acquired sequences of four objects (each consisting of 26 frames) with a commodity RGB-D sensor (Carmine 1.09) to assess the geometrical quality we can achieve with low-cost

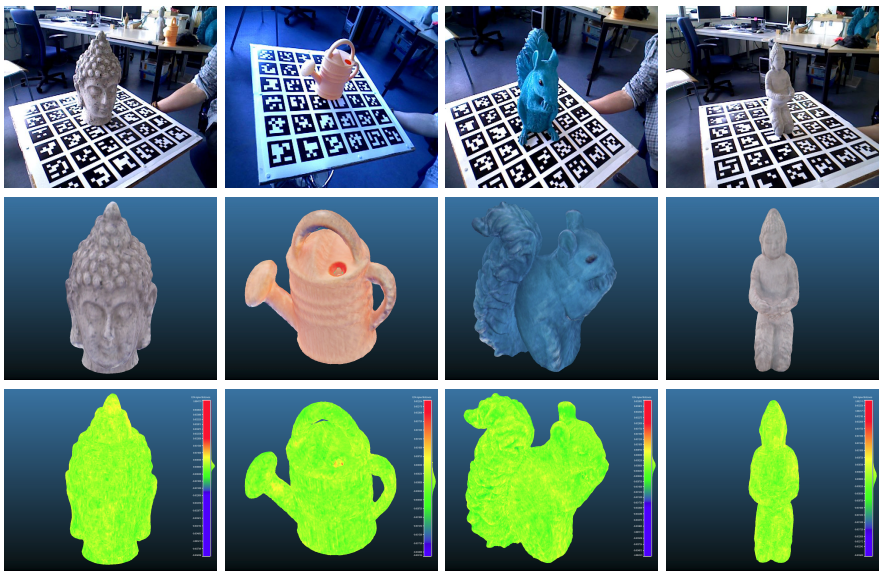


Figure 4.6: The four objects acquired with the Carmine 1.09 sensor (top) together with their dense reconstructions (center) and their vertex-wise difference to their Octree-reconstructed pendants (bottom). The minimum voxel size was set to 1.5mm and $\lambda = 0.3$ for all objects and both methods. The saturation of the difference coloring has been set to $\pm 2mm$. The error induced by our approach stays bounded within the specified voxel size of $1mm$.

devices. Lastly, we acquired 24 frames of a turbine blade taken with an industrial high-precision depth sensor (GIS) that provides micrometer precision. Since we have a CAD model of the turbine we evaluate how accurate we can reconstruct real-life objects with state-of-the-art depth sensing technology which is important for manufacturing verification. For comparisons with dense results, we compare to our method from the last chapter, i.e. [Kehl et al., 2014].

For the experiments we found that running the optimization for 100 iterations with an initial gradient step size $\xi = 0.1$ and halving it every 20 iterations was sufficient to converge to good solutions for any object. Furthermore, we fixed $\eta = 2cm$ but set the metric voxel size s_v and the uncertainty factor δ depending on the data source. For the synthetic data, we set $s_v := 1mm$ and $\delta := 0.1mm$, for the Carmine dataset $s_v := 1.5mm, \delta := 2mm$ and for the GIS data $s_v = 0.5mm, \delta := 0.8mm$.

For the Carmine sequences (Figure 4.6), we constantly retrieve very accurate solutions that are on par with their dense versions. For a more quantitative comparison, we also compare our reconstructions of the 'sphere' and the 'turbine' to their groundtruth models (Figures 4.7 and 4.8). To compute the vertex-wise difference, we find for each vertex of one reconstruction the closest point of the other reconstruction and compute their distance. The synthetic sphere is nicely reconstructed and the mean error of $0.012mm \pm 0.070mm$ is virtually negligible. Also the 'turbine' reconstruction was very accurate with a mean error of $0.22mm \pm 0.50mm$ which is inside tolerable limits.

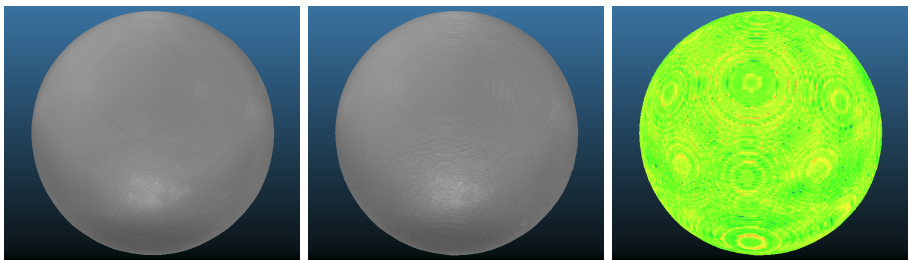


Figure 4.7: Left to right: Dense result, Octree result, Vertex-wise difference.

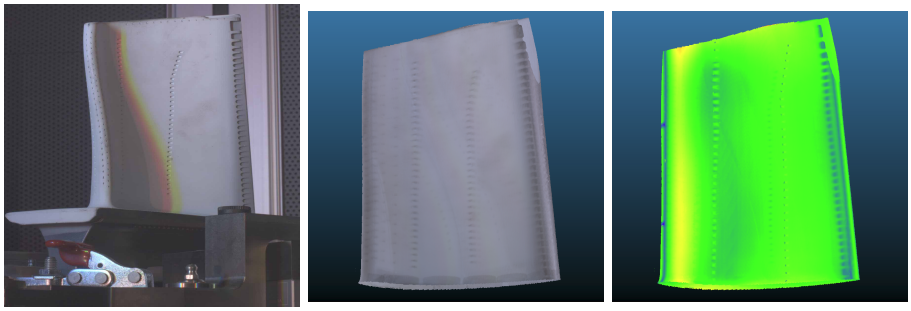


Figure 4.8: 'Turbine' reconstruction. From left to right: One frame from the sequence, the Octree-reconstructed 3D mesh and the difference to the CAD model with mean error of $0.22mm$ and standard deviation of $0.5mm$. Most errors accumulate at the sharp edge on the left as well as the small indents on the right which where smoothed during optimization.

	SDF Size/Res.	Octree Size/Res.
sphere	$3,968$ MB / 256^3	257 MB / 256^3
statue	$3,072$ MB / 256^3	683 MB / 256^3
head	$3,072$ MB / 256^3	806 MB / 256^3
squirrel	$3,072$ MB / 256^3	689 MB / 256^3
can	$3,072$ MB / 256^3	565 MB / 256^3
turbine	$24,576$ MB / 512^3	$2,192$ MB / 512^3

Table 4.1: Memory consumption/volume resolution for the input TSDFs f_i for each sequence.

	Dense	Octree
sphere	3.9	2.8
statue	3.7	2.6
head	3.7	3.4
squirrel	3.7	4.7
can	3.7	3.2
turbine	26.1	9.5

Table 4.2: Time (in minutes) for the optimization.

4.3.1 Memory consumption and runtime

Each dense TSDF stores for each voxel the actual distance value and the weight as floats. In comparison, the Octree-TSDF stores per block pointers to 8 children, its parent node and in addition to the above two floats another float value that represents the distance value just before splitting to allow for the computation of the divergence in the same pass. The total amount of memory needed to hold the data term is given in Table 4.1 and is drastically reduced with the Octree approach for all sequences. To give another interesting insight we plot the memory consumption of u^* during the optimization in Figure 4.9 (left) and show the visual development of the Octree in Figure 4.10. While in the dense approach each iterate u_t is constant in memory, its Octree-variant quickly decreases its footprint after more and more blocks get joined. Analogously for the runtime in Table 4.2, our fast traversing technique allows us to also outperform the dense variants except for the 'squirrel'. While the runtimes for the dense TSDFs are dependent on the volume resolution, the geometry complexity is the main driver of the runtime for the Octree method since frequent repartitionings in-between iterations can lead to a runtime penalty.

4.3.2 Split and join

As stated, we apply split and join rules according to thresholds τ_s and τ_j. Since these values govern the structural density of our Octrees, a careful choice is important to uphold the geometrical accuracy. Splitting early creates a finer partitioning and can lead to unnecessarily high runtime and memory demands while joining too early can result in larger numerical errors since smaller gradient increments get discarded. To have a visual feeling of the impact, we plot some configurations in Figure 4.8 (right) and show the fusion of some Carmine frames taken of a 3D print of the Stanford bunny in Figure 4.10. Apart from the first case that virtually hinders nodes from splitting and leading to massive Octree-artifacts with $\tau_s = 0$, the geometry suffers less from quantization with an increasing τ_s since it steers how fine the Octree describes the area around the narrow band. Conversely, with a higher τ_j we can delay the joining of nodes and thus have the same effect on the area around the band, but from the opposite side, and with a value of $\tau_j > 1$ actually disabling join operations.

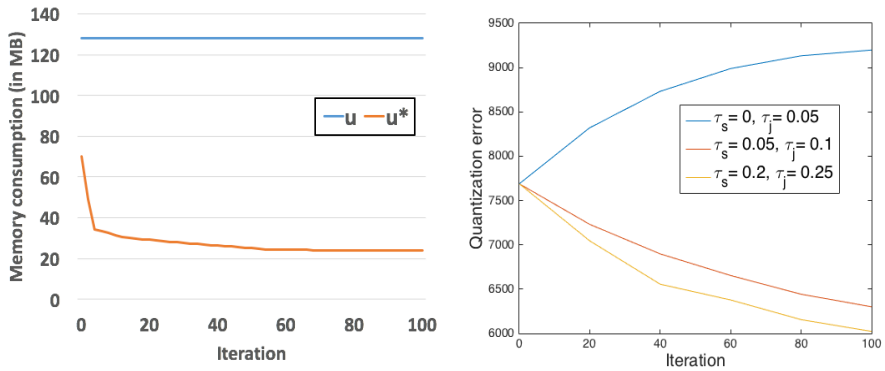


Figure 4.9: Left: Memory usage of the iterate u and u^* during the optimization for the 'head' sequence. The usage goes down quickly for the Octree-variant as the surface evolves in the TSDF, leading to many block joins. Right: The quantization error for three configurations. The higher the thresholds, the smaller the approximation error.

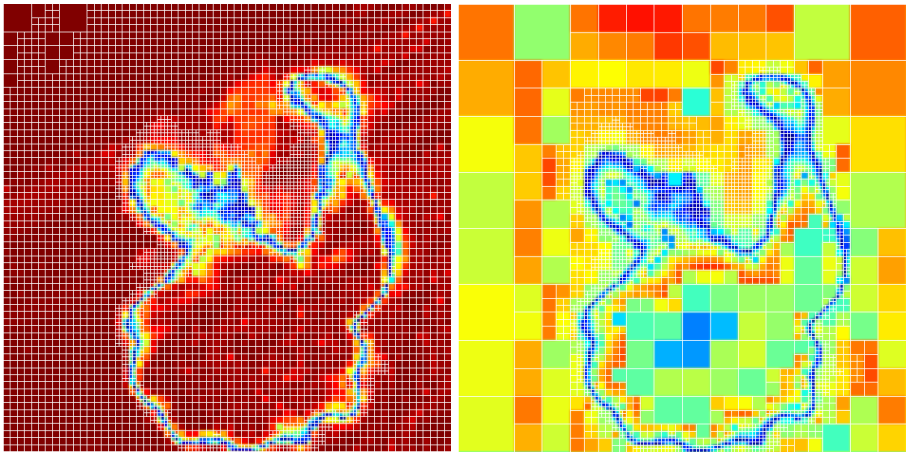


Figure 4.10: Slicing through u^* at iterations 1 and 100.

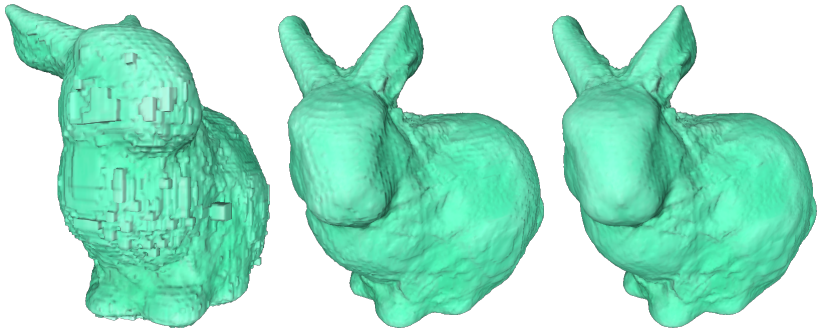


Figure 4.11: The quantization effect of the two join/split thresholds τ_s and τ_j for the three configurations used in the graph above.

The quantization error between a dense u and its Octree-version u^* is

$$\sum_{n \in \text{leafs}(u^*)} \left| u^*(n) - \frac{1}{\Omega_3(n)} \int_{\Omega_3(n)} u(\mathbf{X}) d\mathbf{X} \right| \tag{4.7}$$

and in the ideal case it should be zero for each iterate pair (u_t, u_t^*) during optimization. As can be seen in Figures 4.9 (right) and 4.11 , the error decreases with higher values of both τ_s and τ_j whereas in the special case of $\tau_s = 0$, the error grows larger due to the lack of splitting, leaving the surface heavy with artifacts while the dense version gets smoother.

4.4 Conclusion

We have presented a variational range data fusion approach by partitioning the solutions with a dynamic Octree structure. We have shown how to efficiently conduct restructuring based on iterative node-wise updates of the supplied PDE and that the achieved results are geometrically accurate on multiple datasets and nearly identical to their dense counterparts while being more efficient overall.

Part II

Scalable Detection and Tracking of 3D Objects

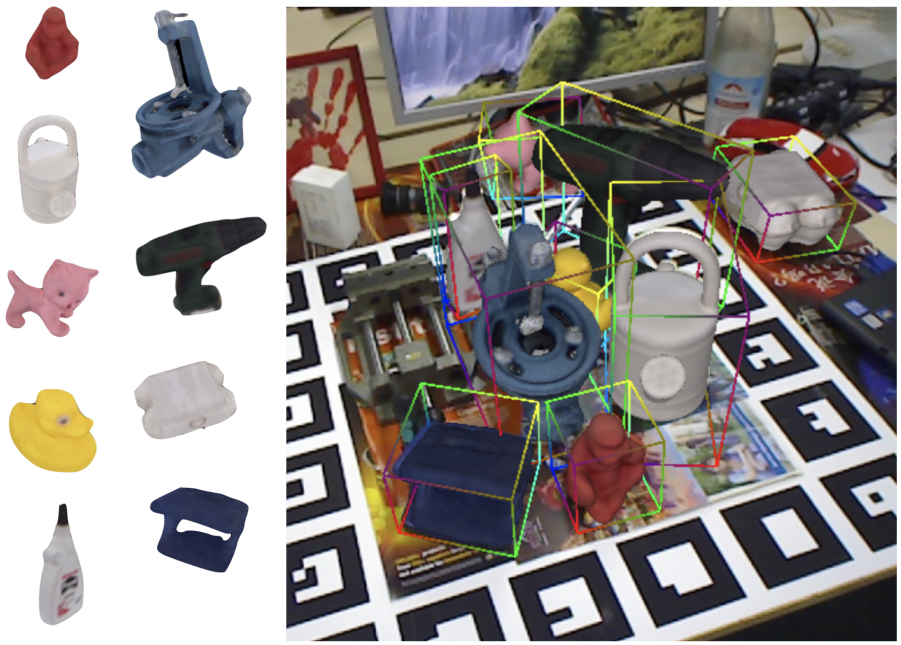


Figure 4.12: 3D models with rendered ground truth poses on one frame from the ACCV12 dataset [Hinterstoisser et al., 2012b]. The challenging task of detection and 6D pose estimation requires reasoning about metric depth, occlusion and rotational symmetry ambiguities.

The second part of the thesis will investigate how to detect known 3D objects in RGB-D data, estimate their 6D poses as well as tracking them over time. Furthermore, we will put a special focus on scalability, i.e. we will explore methods towards dealing with many objects and/or faster processing.

3D detection and 6D Pose Estimation

In essence, we are provided with a set of known 3D objects, usually in the form of colored triangle meshes, and the goal is to

- detect or localize the objects of interest in the image plane
- determine a transformation $\Xi \in SE(3)$ that describes the 6D pose of each object from the camera's point of view.

We refer to Figure 4.12 for a visualization. Note that there are research avenues that explore these problems in other forms of data, e.g. point clouds [Drost et al., 2010, Aldoma et al., 2013, Hodan et al., 2015] or signed distance fields [Tateno et al., 2016], but they are out of scope for this thesis.

While this topic has a long history, starting from simple edge-based methods [Harris and Stephens, 1988, Lowe, 1992], it drastically gained traction with the inventions of the SIFT [Lowe, 2004] and HOG [Dalal and Triggs, 2005] descriptors. Both analyze the image gradient structure either locally or on patches

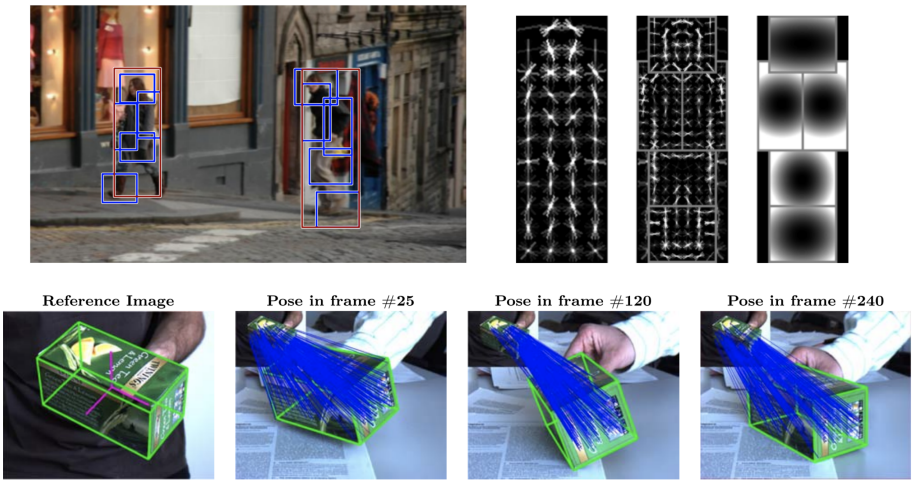


Figure 4.13: Top: Detections from a DPM model with visualized HOG features and spatial weighting, adapted from [Felzenszwalb et al., 2010]. Bottom: Keypoint-based detection and simultaneous 6D pose estimation from [Lepetit et al., 2008].

and enabled (relatively) robust detection performance at that time. While the former relies on nearest-neighbor searches in descriptor space followed by geometrical verification, the latter is applied in a sliding-window fashion, similar to template matching approaches, and feeds the current patch into a previously trained classifier. In fact, the HOG approach is an extension of template matching where the score function has been learned from data instead of manually determined. From a practical standpoint, objects with rich textures can be well described with local descriptive points whereas patch-based gradient information is better suited for poorly-textured objects since they allow better characterization via their contours.

It is important to understand that object detection and pose estimation are both intrinsically coupled for most problems. For example, detecting a car means that we have had access to images of cars in different poses and thus, prediction of the pose can come naturally as an additional quantity with the detection. Both SIFT and HOG have seen many additions and improvements over time and were used for simultaneous detection and pose estimation, e.g. for 6D pose estimation of textured objects with PnP (Perspective-n-Point) algorithms [Lepetit et al., 2008] or for articulated human poses and generally deformable objects in 2D with the DPM model [Felzenszwalb et al., 2010]. See Figure 4.13 for a visualization.

Orthogonally, many researchers investigated ways to increase computational efficiency by either making descriptor matching more intelligent with trees [Nistér and Stewénius, 2006], approximating metrics [Muja and Lowe, 2014] and quantization [Jégou et al., 2011], or by hashing of real-valued descriptors into binary codes [Gong et al., 2013, Lin et al., 2014]. In [Dean et al., 2013] tem-

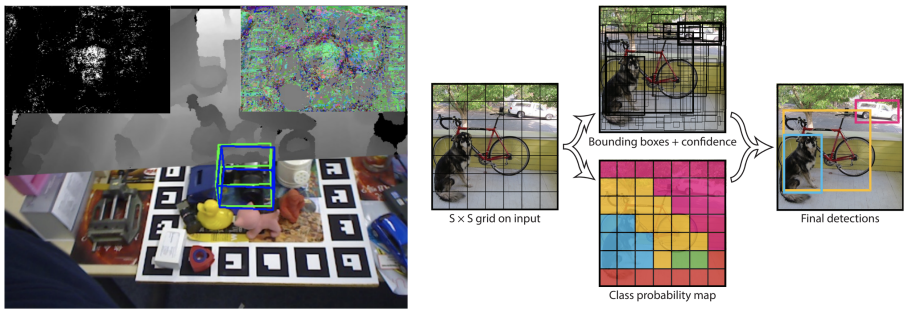


Figure 4.14: Left: Showing the object coordinate vote space together with ground truth and prediction from [Brachmann et al., 2014]. Right: The fully-convolutional detection pipeline from [Redmon et al., 2016].

plate convolutions were replaced by constant-time probing of hash tables in a sliding-window fashion, scaling to 100,000 2D object classes. In a similar way, [Aytar and Zisserman, 2014] present a scalable detector by representing HOG sparsely with a set of patches which can be retrieved immediately.

Lending from the HOG idea, detection and pose estimation approaches based on view-dependent 2D object contours for 3D estimation then became popular. In [Ulrich et al., 2012] the authors explore a visual aspect graph to cluster similar view contours into a hierarchical pose tree. [Damen and Bunnun, 2012] hashes paths over edgelets in color images and allows for real-time 3D pose detection, however the output remains in terms of 2D locations only. [Cai et al., 2013] applies uniform quantization to edge-based descriptors to immediately look up approximate nearest neighbors. LineMOD [Hinterstoisser et al., 2012a] achieved robust 3D object detection and pose estimation by efficiently matching templated views with quantized object contours and normal orientations, which were computed from the depth channel. In [Rios-Cabrera and Tuytelaars, 2013] the authors further optimize the matching via cascades and fine-tuned templates. The next chapter will take on this avenue and show possible improvements by learning hashing functions on these templates that include pose space information.

Around this time, methods that learn representations started to show better performance than those based on hand-crafted features. [Brachmann et al., 2014, Tejani et al., 2014] use random forest-based voting schemes on local patches to detect and estimate 6D poses. While the former regresses object coordinates and conducts a subsequent energy-based pose estimation (see Figure 4.14, left), the latter bases its voting on a scale-invariant patch representation and returns location and pose simultaneously. In chapter 6 we will present a novel formulation that replaces random forests with descriptors learned from a deep model and, together with approximate nearest-neighbor matching, providing a scalable alternative.

In parallel, CNN (convolutional neural network) models such as R-CNN [Girshick et al., 2014] and its variants outperformed traditional 2D object detectors by a large margin. Their idea is to first select regions of interest (ROI)

in the image and then feed those regions to a CNN that classifies them. Later versions, such as Faster R-CNN [Ren et al., 2015] and especially the advent of YOLO [Redmon et al., 2016] and SSD [Liu et al., 2016], [Lin et al., 2017] enabled real-time detection at high accuracy and practically deprecated all previous approaches. Their idea is to inverse the sampling strategy such that scene sampling is not anymore a set of discrete input sample points leading to continuous output. Instead, the input space is dense on the whole image and the output space is discretized into many overlapping bounding boxes of varying shapes and sizes. This inversion allows for smooth scale search over many differently-sized feature maps and simultaneous classification of all boxes in a single pass. In order to compensate for the discretization of the output domain, each bounding box regresses a refinement of its corners. We refer to Figure 4.14 for a visualization of YOLO. We will present in chapter 7 an extension of these models to full 6D pose estimation that is able to outperform previous methods while being much faster. More recent methods have adopted this paradigm as well and learned to regress projected cuboid points followed by PnP [Rad and Lepetit, 2017, Tremblay et al., 2018] or regress the pose properties from ROI-windowing mechanisms inside the network that focus on each instance [Xiang et al., 2018, Manhardt et al., 2019b].

6D Pose Tracking

In respect to 6D model tracking, the formulation of the problem changes since tracking usually assumes an initialization. Therefore, we are given a 3D model together with an initial pose estimate $\Xi_{t=0}$ and the task is to estimate the pose Ξ_{t+1} at the next time step from the input data. This is quite challenging since objects can be ambiguous in their pose and can undergo occlusions as well as appearance changes. Furthermore, trackers must also be fast enough in order to cover larger inter-frame motions.

In the case of pose tracking from color images, earlier works employ either 2D-3D correspondences [Rosenhahn et al., 2006, Schmaltz et al., 2007] or 3D edges [Drummond and Cipolla, 2002, Tateno et al., 2009, Seo et al., 2014] and fit the model in an ICP fashion, i.e. without explicitly computing a contour. While successive methods along this direction managed to obtain improved performance [Brox et al., 2010, Schmaltz et al., 2012], another set of works solely focused on tracking densely the contour by evolving a level-set function [Bibby and Reid, 2008, Dambreville et al., 2010]. As a particular work, [Bibby and Reid, 2008] aligned the current evolving 2D contour to a color segmentation, and demonstrated improved robustness when computing a posterior distribution in color space. In [Prisacariu and Reid, 2012], the contour is determined and tracked by projecting the 3D model with its associated 6D pose onto the frame. Then, the alignment error between segmentation and projection drives the update of the pose parameters via gradient descent. In [Prisacariu et al., 2015], the authors extend their method to simultaneously track and reconstruct a 3D object on a mobile phone in real-time. They circumvent GPU rendering by hierarchically ray-casting a volumetric representation and speed up pose optimization by exploiting the phone's inertial sensor data. [Tjaden et al., 2016] built on the original framework and extended

it with a new optimization scheme together with a twist representation. Additionally, they handle occlusions in a multi-object tracking scenario, making the whole approach more robust in practice. The typical problem of these methods is their fragile segmentation based on color histograms, which can fail easily without using an adaptive appearance model, or when tracking in scenes where the background colors match the objects' colors. Based on this, [Zhao et al., 2014] explores a boundary term to strengthen contours, whereas [Hexner and Hagege, 2016, Tjaden et al., 2019] improve the segmentation with local appearance models.

For temporal tracking in depth, most works employ energy minimization [Choi and Christensen, 2013, Ren et al., 2014, Slavcheva et al., 2016]. While the method from [Choi and Christensen, 2013] uses a particle filter approach for hypothesis sampling, [Ren et al., 2013, Ren et al., 2014] simultaneously track and reconstruct a 3D level-set embedding from depth data, following a color-based segmentation.

About the same time, learned approaches for tracking started with Random Forests [Tan and Ilic, 2014, Tan et al., 2015, Krull et al., 2014] and were complemented by deep models [Manhardt et al., 2018, Li et al., 2018] soon after. While the former works simply learn split functions on sample positions to produce discretized pose increments, the latter learned continuous updates on dense images. The advantage of the deep models is the possibility of pretraining on unrelated images beforehand, and leveraging the rich feature hierarchies for better generalization. In fact, those two works have shown that the learned tracking method can even generalize to categories of objects.

We will summarize the context of the following chapters of this part before giving in-more detailed explanations in their respective sections.

Hashing of Multi-Modal Binary Descriptors The fourth chapter is based on *Hashmod: A Hashing Method for Scalable 3D Object Detection* (BMVC2015). Here we will present a scalable method for detecting objects and estimating their 6D poses in RGB-D data. To this end, we rely on an efficient representation of object views and employ hashing techniques to match these views against the input frame in a scalable way. While a similar approach already exists for 2D detection, we show how to extend it to estimate the 6D pose of the detected objects. In particular, we explore different hashing strategies and identify the one which is more suitable to our problem. We show empirically that the complexity of our method is sublinear with the number of objects and we enable detection and pose estimation of many 3D objects with high accuracy and fast runtime.

Convolutional Autoencoders for 6D Instance Voting The fifth chapter is based on *Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation* (ECCV2016). We will discuss a 3D object detection method that uses regressed descriptors of locally-sampled RGB-D patches for 6D vote casting. For regression, we employ a convolutional auto-encoder that

has been trained on a large collection of random local patches. During testing, scene patch descriptors are matched against a database of synthetic model view patches and cast 6D object votes which are subsequently filtered to refined hypotheses. We evaluate on three datasets to show that our method generalizes well to previously unseen input data, delivers robust detection results and is scalable in the number of objects.

Single-Shot Detection for 6D Pose Estimation The sixth chapter is based on *SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again* (ICCV2017), which was selected for oral presentation. The first authorship of this publication is shared with Fabian Manhardt, who contributed significantly to the implementation and evaluation of the work. In essence, we present a novel method for detecting 3D model instances and estimating their 6D poses from RGB data in a single shot. To this end, we extend the popular SSD paradigm to cover the full 6D pose space and train on synthetic model data only. Our approach competes with or surpasses methods that leverage RGB-D data on multiple challenging datasets. Furthermore, our method produces these results at around 10Hz, which is many times faster than the related methods.

Tracking via Joint Contour and Cloud Information The seventh chapter is based on *Real-Time 3D Model Tracking in Color and Depth on a Single CPU Core* (CVPR2017). We will present a novel method to track 3D models in color and depth data. To this end, we introduce approximations that accelerate the state-of-the-art in region-based tracking by an order of magnitude while retaining similar accuracy. Furthermore, we show how the method can be made more robust in the presence of depth data and consequently formulate a new joint contour and ICP tracking energy. We present better results than the state-of-the-art while being much faster then most other methods and achieving all of the above on a single CPU core.

Chapter 5

Hashing of Multi-Modal Binary Descriptors

In this chapter, our approach to 3D object detection will be based on 2D view-specific templates which cover the appearance of the objects over multiple viewpoints [Hoiem and Savarese, 2011, Nayar et al., 1996, Gu and Ren, 2010, Hinterstoisser et al., 2012b, Aubry et al., 2014]. Since viewpoints include the whole object appearance rather than just parts of it, they can generally handle objects with poor visual features.

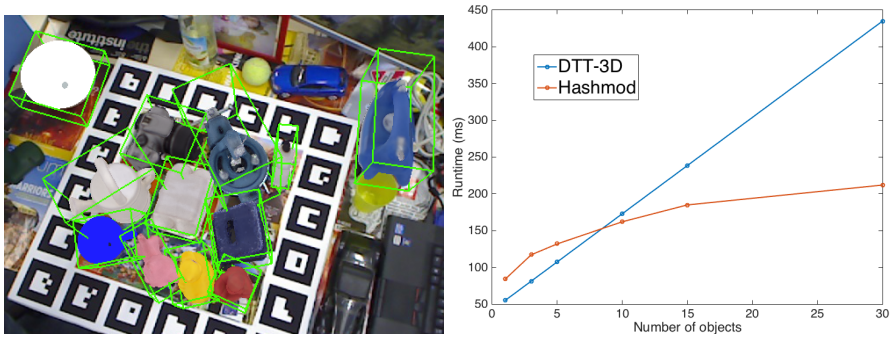


Figure 5.1: Left: One frame of the ACCV12 dataset [Hinterstoisser et al., 2012b] augmented with our detections. Right: Average performance of our approach with a given amount of objects in the database. We scale sublinearly and outperform DTT-3D with more than 8 objects, enabling detection of many 3D objects at interactive runtimes.

We apply hash functions [Gionis et al., 1999] to image descriptors computed over bounding boxes centered at each image location of the scene, so to match them efficiently against a large descriptor database of model views. In our work, we rely on the LineMOD descriptor [Hinterstoisser et al., 2012b], since it has been shown to work well for 3D object detection.

Our contribution is to present an efficient way of hashing such a descriptor:

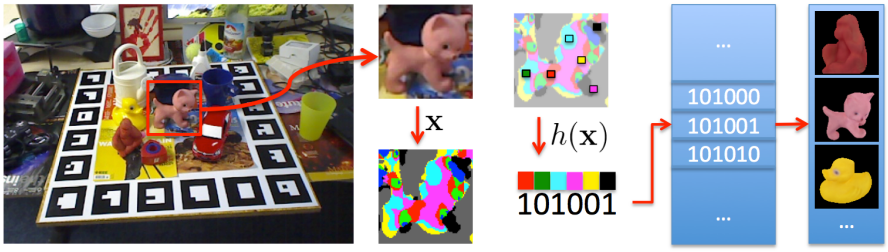


Figure 5.2: Visualization of the hashing pipeline with one hash function h and a key length of $b = 6$. At each sliding window's position we extract a LineMOD descriptor $\mathbf{x}$ and sample certain orientations at specific positions to form a short binary string $h(\mathbf{x})$. This serves as an index into the prefilled hash table to retrieve candidate views for further matching. Both the sample positions and orientations for h are learned.

to this end, we explore different learning strategies to identify the one that is most suited to our problem. As shown in Figure 5.1, we outperform the state-of-the-art template matching method DTT-3D [Rios-Cabrera and Tuytelaars, 2013] by intelligently hashing our descriptors, achieving sublinear scalability.

5.1 Construction of Binary Descriptors

Given a database of M objects, we synthetically create N views for each object from poses regularly sampled on a hemisphere of a given radius, as shown in Figure 5.3. From this, we compute a set $\mathcal{D}$ of d-dimensional binary descriptors:

$$\mathcal{D} = \{\mathbf{x}_{1,1}, ..., \mathbf{x}_{M,N}\} \quad , \tag{5.1}$$

where $\mathbf{x}_{i,j} \in \mathbb{B}^d$ is the descriptor for the i-th object seen under the j-th pose. As already mentioned, we use LineMOD in practice to compute these descriptors. The LineMOD descriptor is a vector of integers between 0 and 16 and for each pixel it is either set to 0 when there is no significant image gradient or depth data present or otherwise set to a value to represent a quantized orientation of either an image gradient (1-8) or 3D normal (9-16). We concatenate the binary representation of these integer values to obtain the binary strings $\mathbf{x}_{i,j}$. In the remainder of this work we will use the terms views, templates, and descriptors as synonyms.

Figure 5.2 gives an overview of our pipeline. As usually done in template-based approaches, we parse the image with a sliding window looking for the objects of interest. We extract at each image location the corresponding descriptor $\mathbf{x}$. If the distance between $\mathbf{x}$ and its nearest neighbor $\mathbf{x}_{i,j}$ in $\mathcal{D}$ is small enough, it is very likely that the image location contains object i under pose j. As discussed in the introduction, we want to perform this (approximate) nearest neighbor search by hashing the descriptors. Therefore, we explore different strategies for building the hashing functions. This is done in the offline stage described below.

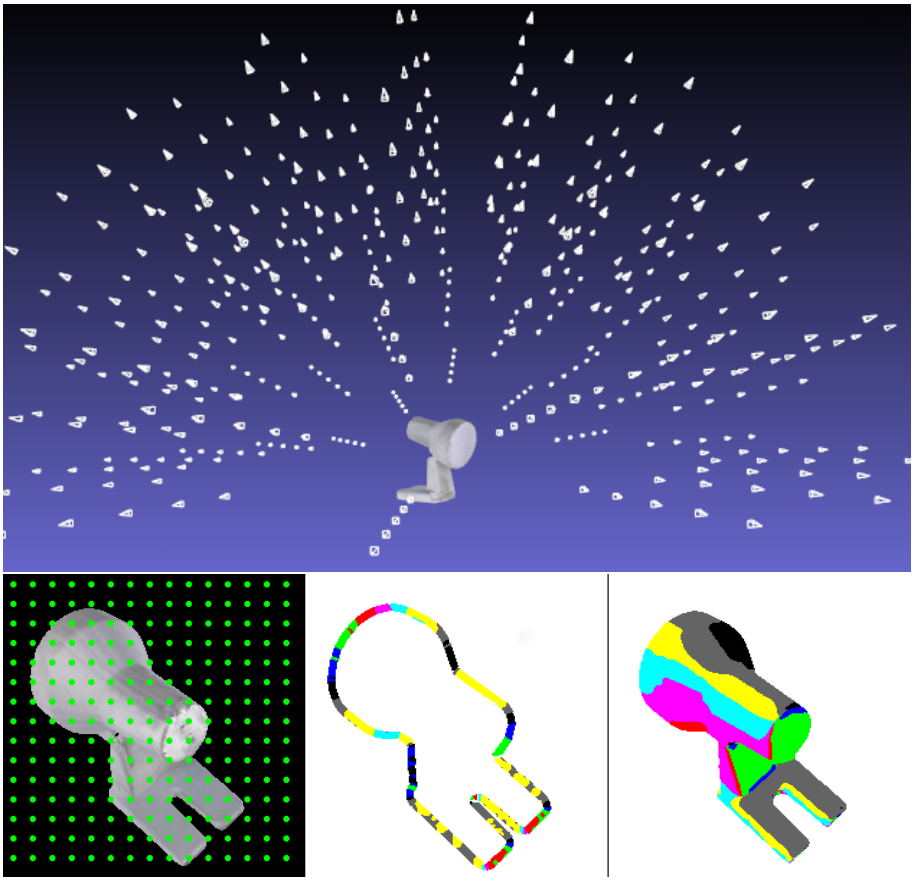


Figure 5.3: Top: As in [Hinterstoisser et al., 2012b], we compute LineMOD descriptors for synthetically rendered views sampled on hemispheres of several radii. Bottom: One such synthetic view of the 'lamp' object overlaid with the fixed grid for matching and the color-coded quantized orientations of image gradients and 3D normals used to compute the descriptor.

We tackle the issue of object scale and views of different 2D sizes by dividing the views up into clusters $\mathcal{D}_s \subset \mathcal{D}$ of similar scale s. This leads to s differently-sized sliding windows during testing which extract differently-sized descriptors on which to perform the hashing. Moreover, to increase detection rates, we assign a pre-defined number of hash functions per window such that they relate to random but overlapping subsets of $\mathcal{D}_s$. During testing, we evaluate all sliding windows with their associated hashing functions, union their retrieved views and conduct subsequent matching. Lastly, we determined a good compromise for the key lengths by setting for each scale s the key length $b_s := \lfloor \log_2(|\mathcal{D}_s|) \rfloor$.

5.2 Selecting the Hashing Keys

During our offline stage, we learn several hashing functions h [Gionis et al., 1999]. As shown in Figure 5.2, the purpose of each function is to immediately index into a subset, often called a "bucket", of $\mathcal{D}$ when applied to a descriptor $\mathbf{x} \in \mathbb{B}^d$ during testing. These buckets are filled with descriptors from $\mathcal{D}$ with the same hash value so that we can restrict our search for the nearest neighbor of $\mathbf{x}$ to the bucket retrieved via the hashing function instead of going through the complete set $\mathcal{D}$. It is very likely, but not guaranteed, that the nearest neighbor is in at least one of the buckets returned by the hashing functions.

In practice, a careful selection of the hashing functions is important for good performance. Since the descriptors $\mathbf{x}$ are already binary strings, we design our hashing functions $h(\mathbf{x})$ to return a short binary string made of b bits directly extracted from $\mathbf{x}$. This is a very efficient way of hashing and we will refer to these short strings as hash keys.

There is a typical trade-off between accuracy and speed: we want to retrieve only a handful of descriptors at each image location and the number of retrieved elements is governed by the hash key length and the distribution of the descriptors among the buckets. Since we use b bits for the hash table, we span a table with 2^b buckets. If a key is too short, the number of buckets is too small and we store over-proportionally many descriptors per bucket, increasing subsequent matching time after retrieval. If the key is too large, we might be more prone to noise in the bitstrings which may lead to wrong buckets, rendering false negatives more probable during testing. We thus want to select these b bits in a way such that we maximize the odds of finding the nearest neighbors of the input descriptors in the buckets while keeping the total amount of retrieved views to a minimum.

An exhaustive evaluation of all the possible bit selections to build the hash keys is clearly intractable. We experimented with the following variants:

5.2.1 Randomness-based selection (RBS)

Given a set of descriptors, we select the b bits randomly among all possible d bits. As we will see later on, this selection strategy yields bad results since some bits are more discriminant than others in our template representation. Nonetheless, it provides us with a weak baseline we can compare to and it outlines the importance of a more sophisticated approach towards hash learning.

5.2.2 Probability-based selection (PBS)

For this strategy, we focus on the bits for which the probabilities of being 0 and 1 are close to 0.5 with a given set of descriptors. We therefore rank each bit B according to its entropy $E = p(B = \texttt{0}) \ln p(B = \texttt{0}) + p(B = \texttt{1}) \ln p(B = \texttt{1})$ and take the best b. This strategy provides a high accuracy since it focuses on the most discriminant bits. However, later evaluation will reveal that this strategy results in a high variance in the number of elements per bucket, rendering PBS inefficient in terms of runtime.

5.2.3 Tree-based selection (TBS)

This strategy is inspired by greedy tree growing for Randomized Forests. Starting with a set of descriptors at the root, we determine the bit that splits this set into two subsets with sizes as equal as possible, and use it as the first bit of the key. For the second bit, we decide for the one that splits those two subsets further into four equally-sized subsets and so forth. We stop if b bits have been selected or one subset becomes empty. This procedure alone yields a balanced tree with leafs of similar numbers of elements. Each hash key can be regarded as a path down the tree and each leaf represents a bucket. Note that such a balanced repartition ensures retrieval and matching at a constant speed. Formally, the j-th bit B of the key is selected by solving:

$$\arg\min_{B} \sum_{i} \left| |\mathcal{S}_L^B(N_i)| - |\mathcal{S}_R^B(N_i)| \right| \ , \tag{5.2}$$

where $N_i \subset \mathcal{D}$ is the set of descriptors contained by the i-th node at level j, and $\mathcal{S}^B_{\{L,R\}}(N_i)$ are the two subsets of N_i that go into the left and right child induced by splitting with B.

5.2.4 Tree-based selection with view scattering (TBV)

We now further adapt the TBS strategy to our problem: as illustrated in Figure 5.4, to improve detection rates we favor similar views of the same object to go into different branches. The idea behind this strategy is to reduce misdetections due to noise or clutter in the descriptor. If an extracted hash has a polluted bit and thus points to a wrong bucket, we might not retrieve the best view but still could recover from a similar view that we stored in the bucket the polluted hash points to. This strategy improves the robustness of the TBS retrieval, resulting in a consistently higher recall. We optimize the previous criterion with an additional term:

$$\arg\min_{B} \frac{1}{|N_i|} \sum_{i} \left| |\mathcal{S}_L^B(N_i)| - |\mathcal{S}_R^B(N_i)| \right| + \frac{1}{|N_i|^2} \Big(P(\mathcal{S}_L^B(N_i)) + P(\mathcal{S}_R^B(N_i)) \Big) \ , \tag{5.3}$$

where the second term penalizes close views falling into the same side of the split. We define the penalty function $P(\cdot)$ as:

$$P(N) := \sum_{\mathbf{x} \in N} \sum_{\mathbf{y} \in N} \mathbb{I}(\mathbf{x}, \mathbf{y}) \cdot \begin{cases} 1, \text{ if } \cos^{-1}(|\langle q_\mathbf{x}, q_\mathbf{y} \rangle|) < \tau \\ 0, \text{ otherwise } , \end{cases} \tag{5.4}$$

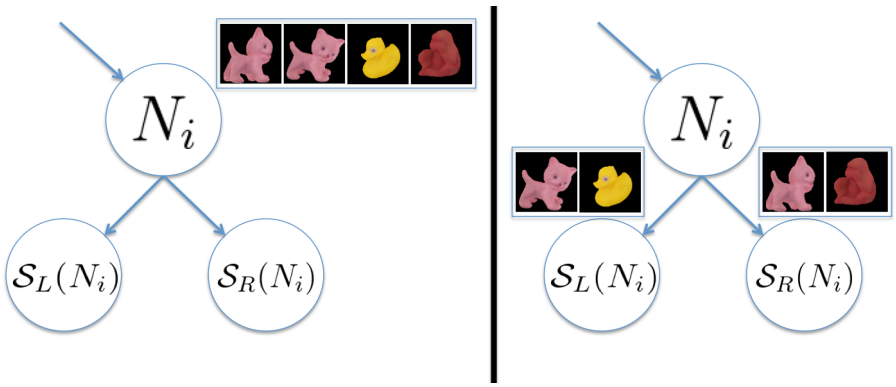


Figure 5.4: The TBV strategy encourages descriptors for similar views to fall into different buckets. This increases the chances to find a close descriptor when parsing the buckets.

where $\mathbb{I}(\mathbf{x}, \mathbf{y})$ indicates if descriptors $\mathbf{x}$ and $\mathbf{y}$ encode views of the same object and $q_\mathbf{x}, q_\mathbf{y}$ are the quaternions associated with the rotational part of the descriptors' poses. We set the proximity threshold $\tau = 0.3$ empirically according to our viewpoint sampling.

5.3 Remarks on the Implementation

For selecting the hash keys, we rely on the descriptors after 'bit spreading' of LineMOD [Hinterstoisser et al., 2012b], which makes the descriptors robust to small translations and deformations in a neighborhood of T pixels. It increases the spatial overlap of quantized features and allows for a better descriptor separability. For matching itself we used the unspreaded templates.

Furthermore, after one bit has been selected, we disallowed all bits closer than T to be selected for the same LineMOD value. This forces the bit selection to take different values and positions into account, as we sometimes observed an accumulation of selected bits encoding the same orientation in one area which could lead to bad recognition rates.

We conduct the matching analogously to [Rios-Cabrera and Tuytelaars, 2013] efficiently on a fixed grid. As opposed to [Hinterstoisser et al., 2012b], we do not use a robust cosine-based similarity score but count the bits after ANDing the descriptors and dividing by the number of grid points falling on the view foreground.

5.4 Evaluation

We ran our method on the ACCV12 dataset [Hinterstoisser et al., 2012b] consisting of 15 objects and followed the exact same protocol to create an equidistant viewpoint sampling. Furthermore, scale and in-plane rotations were sampled accordingly to cover a predefined 6D pose space, resulting in exactly

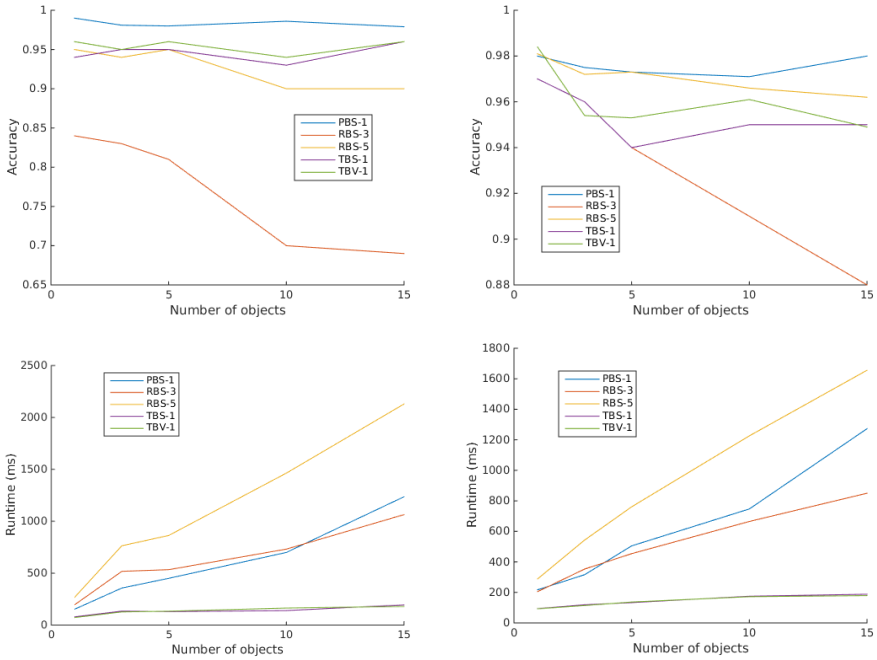


Figure 5.5: Examples of accuracies and runtimes for different strategies and a varying database size. Left column: for the 'ape' sequence. Right column: for the 'lamp' sequence. The number in the legend for each strategy denotes the amount of hash keys per window.

$N = 3115$ views per object.

We followed [Hinterstoisser et al., 2012b] and spread the quantized values in a small neighborhood of $T = 8$ pixels which makes the representation robust and allows to check only every T-th image position. Furthermore, we use the same post-processing: after retrieval/matching, we sort the candidates according to their score and run a rough color check to discard obvious outliers. We conduct a fast voxel-based ICP [Fitzgibbon, 2001] and reject candidates if the average euclidean error is too large. Finally, the first $n = 10$ survivors are projected onto the scene to run a finer ICP together with a depth check to decide for the best match. We use the same evaluation criteria with a distance factor of $k_m = 0.1$ to decide for a hit or miss.

The computational cost during testing is modest: the whole system runs on a single CPU-core—apart from the post-processing where the depth check projections use OpenGL calls—, uses no pyramid scheme and the hash tables take up less than 1 MB.

5.4.1 Different learning strategies.

We learned the hashing for each sequence and object configuration by grouping the object of interest together with a random subset of the remaining ones. An

Seq / Objs	1	3	5	10	15
ape	96.1% / 73ms	95.1% / 127ms	96.8% / 134ms	94.1% / 164ms	95.6% / 180ms
bvise	92.8% / 83ms	95.4% / 117ms	91.2% / 128ms	92.3% / 181ms	91.2% / 192ms
bowl	99.3% / 84ms	98.8% / 114ms	98.6% / 123ms	98.1% / 156ms	98.6% / 184ms
cam	97.8% / 75ms	96.9% / 111ms	95.3% / 129ms	94.8% / 162ms	95.2% / 174ms
can	92.8% / 81ms	91.6% / 112ms	93.7% / 119ms	93.9% / 158ms	91.8% / 171ms
cat	98.9% / 99ms	97.2% / 117ms	97.4% / 138ms	95.8% / 164ms	96.1% / 188ms
cup	96.2% / 65ms	96.0% / 100ms	95.3% / 117ms	94.5% / 144ms	98.6% / 194ms
driller	98.2% / 106ms	97.8% / 135ms	98.1% / 162ms	97.6% / 171ms	95.1% / 190ms
duck	94.1% / 74ms	90.8% / 122ms	90.7% / 124ms	91.5% / 161ms	92.9% / 179ms
eggbox	99.9% / 68ms	99.9% / 103ms	99.9% / 115ms	99.9% / 151ms	99.9% / 174ms
glue	96.8% / 100ms	96.2% / 131ms	94.4% / 154ms	95.3% / 166ms	95.4% / 175ms
hpuncher	95.7% / 97ms	95.3% / 118ms	95.8% / 142ms	95.2% / 162ms	95.9% / 183ms
iron	96.5% / 101ms	94.8% / 122ms	95.0% / 141ms	95.5% / 167ms	94.3% / 203ms
lamp	98.4% / 93ms	95.4% / 114ms	95.3% / 137ms	96.1% / 172ms	94.9% / 179ms
phone	93.3% / 90ms	94.6% / 126ms	94.5% / 133ms	91.7% / 167ms	91.3% / 198ms
TBV Average	96.5% / 83ms	95.7% / 117ms	95.5% / 131ms	94.9% / 162ms	95.1% / 184ms
DTT-3D	97.2% / 55ms	97.2% / 81ms	97.2% / 107ms	97.2% / 173ms	97.2% / 239ms
LineMOD	96.6% / 119ms	96.6% / 273ms	96.6% / 427ms	96.6% / 812ms	96.6% / 1197ms

Table 5.1: Accuracy and runtime per frame for our whole pipeline with the TBV strategy, DTT-3D and LineMOD for the whole dataset with a varying number of trained and loaded objects. With only a few objects, DTT-3D is faster than our approach. However, its complexity increases linearly with the database size, allowing us to overtake when the number of objects becomes higher. Note that the dataset only provides groundtruth for one object per frame.

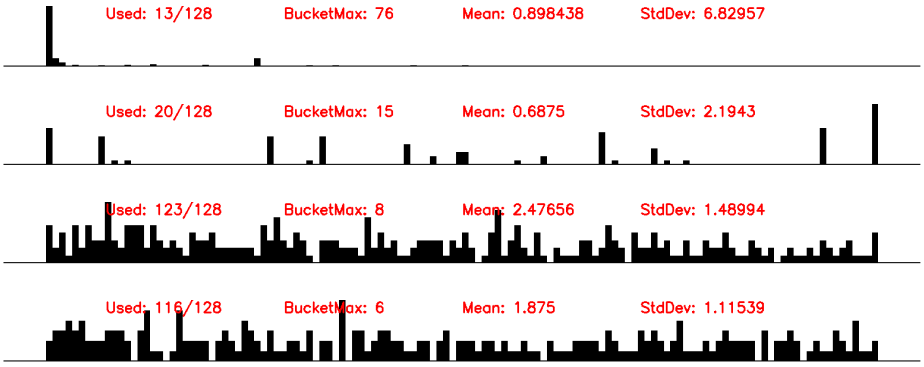


Figure 5.6: Bucket distribution for key length $b = 7$ and different strategies on 'benchvise'. From top to bottom: RBS, PBS, TBS, TBV. Note that the height for each hash table is normalized, skewing the comparison optically.

exception is the case for 15 objects where we built the hash tables once and used them for all tests. A summary of our evaluation is given in Figure 5.5. Note that we conducted our experiments with a varying amount of hash tables per window/scale for each strategy but only plot the most insightful to not clutter the graphs and save space. The behavior was similar across the whole dataset and we thus present results for all strategies only on two sequences and then restrict ourselves to the best strategy thereafter for a more detailed analysis.

The RBS strategy was clearly the weakest one. This is because RBS managed a poor separation of descriptors: since the key bits were chosen randomly, most descriptors were assigned a hash value of pure zeros and were put into the first bucket while the rest of the hash table was nearly empty. This resulted during testing in either hitting an arbitrary bucket with no elements or the 0-bucket with a high amount of retrieved views, approximating an exhaustive search at that image location which increased matching time. It only started to detect accurately with multiple tables per window at the expense of very high runtimes.

Not surprisingly, PBS nearly always managed to correctly detect the object—limited only by our matching threshold—while achieving similar runtimes as RBS with 3 tables per window. An inspection of the hashes revealed that PBS led to multiple large buckets where descriptors concentrated and if one of those buckets was hit, it was very likely that it contained the correct view. Nonetheless, PBS does not take advantage of all available buckets as some of them remain empty and therefore still exhibits a linear runtime growth. Using more tables for RBS was just slightly increasing runtime and accuracy.

For both TBV and TBS the most interesting observation is their sublinear growth in runtime. Enforcing the tables to be filled equally results in an obvious drop in the amount of retrieved views. Nonetheless, both strategies yield already good accuracies with one table per window and TBV was able to outperform TBS usually with around 2% in accuracy since otherwise missed views could be retrieved and ICP-refined to the same correct pose from a similar view in another bucket.

In Figure 5.6 we present actual instantiations of the hash tables for a key

length of $b = 7$ on the 'benchvise' sequence, providing us with tables of 128 possible buckets for each strategy. We provide 3 important quantities: the actual number of used buckets, the largest number of descriptors that is in one bucket as well as the standard deviation computed over all non-empty buckets.

Apparently, RBS produced by far the largest disparity in bucket sizes where the largest, storing 76 descriptors, is also at the same time the mentioned 0-bucket. The remaining used 12 buckets are rather small and the other 115 are completely empty, leading to a high standard deviation. Hitting the 0-bucket retrieves a lot of descriptors, shooting up runtime unnecessarily while otherwise the method retrieves virtually nothing at all when hitting any other bucket.

PBS already uses more buckets than RBS and creates a more balanced separation into multiple larger sets. The best results are however obtained with our tree-based selection strategies that provide us with a good distribution among the buckets, i.e. small bucket sizes together with a small standard deviation. In the optimal case, the standard deviation would be zero and the buckets' usage would equal an uniform distribution.

5.4.2 Different number of tables per scale

We evaluated each strategy with a different amount of trained hash tables per scale. The descriptors for each scale were subdivided into multiple overlapping groups and then one table was responsible for indexing into one of them. As we can observe in Figure 5.7, increasing the amount of tables always leads to better recognition at the expense of higher runtimes. Except for the RBS strategy which would have needed even more tables still, all methods experienced a saturation in detection accuracy for the 'ape' around 99.5%. In light of to these results, we settled for TBV and one table per scale because it allowed us to be faster than DTT-3D with a comparable accuracy.

5.4.3 Comparison to related methods.

Since TBV supplies us with a sublinear runtime growth and acceptably high accuracies, we settle for this strategy and show more detailed results in Table 5.1. We are able to consistently detect at around $95\% - 96\%$ accuracy on average which is slightly worse than LineMOD and DTT-3D. However, we are always faster than LineMOD and overtake DTT-3D at around 8 objects where our constant-time hashing overhead becomes negligible and the methods' time complexities dominate. This is important to stress since real scalability comes from a sublinear growth. Additionally, we show more clearly the scalability of our approach when increasing the amount of descriptors: since the dataset consists of only 15 objects, yielding 46,725 descriptors, we created further 46,725 descriptors by drawing each bit from its estimated distribution, thus enlarging our database artificially to 30 objects. Figure 5.1 shows a graph of our runtimes in comparison to DTT-3D. For the latter, we extrapolated the values given the authors' timings. The gap in runtime shows our superiority when dealing with many objects and views.

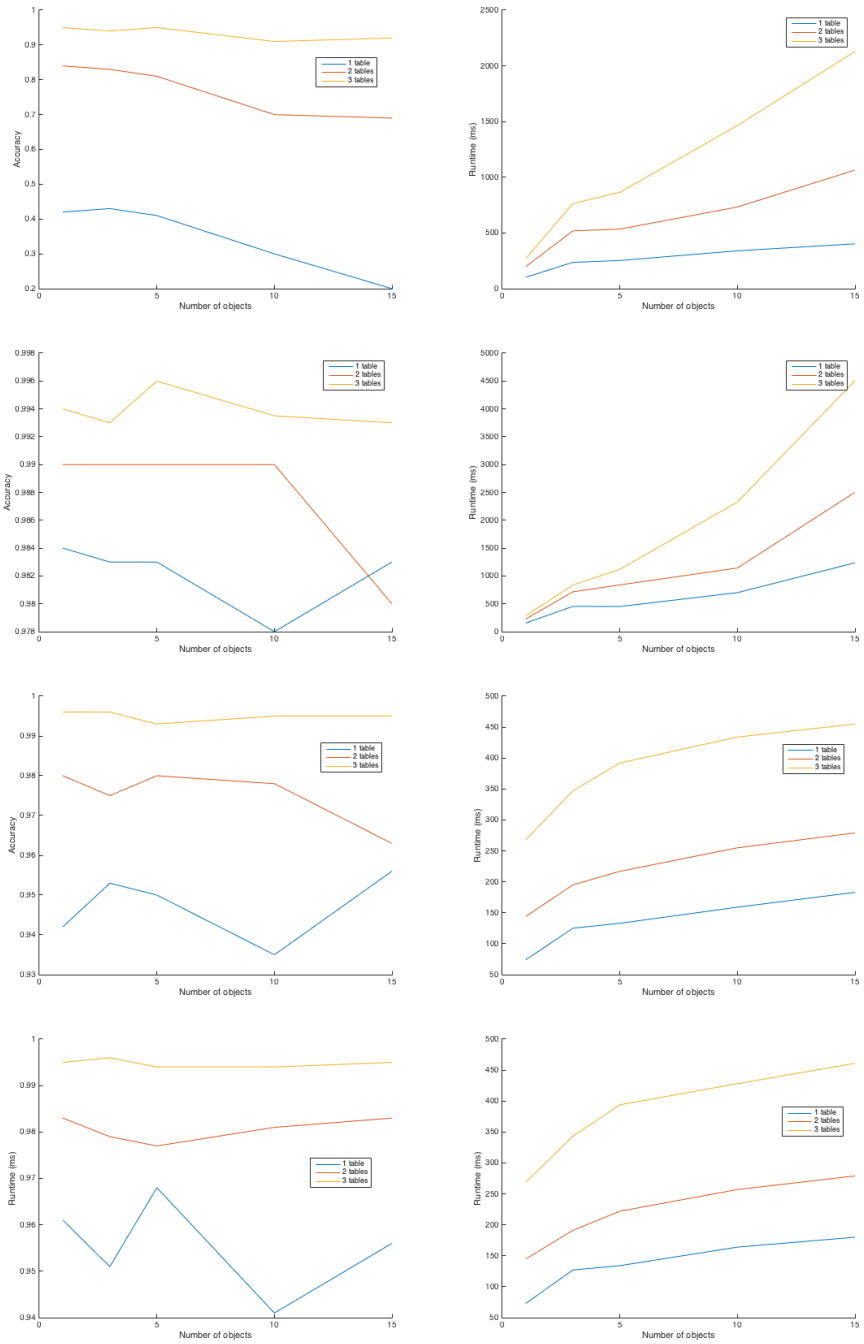


Figure 5.7: The behavior of each strategy on the 'ape' sequence when the amount of tables per scale increases. From top to bottom: RBS, PBS, TBS, TBV.

5.4.4 Sublinear retrieval and matching.

After retrieval, we conduct template matching together with an object-dependent threshold. Although this parameter is of importance to balance runtime versus accuracy, we are less prone to ill settings in comparison to LineMOD since at each position we only retrieve a tiny subset of candidates, as shown in Figure 5.8 (left). Obviously, the small set of retrieved views most often contains the correct one, leading to good accuracies while keeping the runtime low. Furthermore, the ratio of total conducted matchings on an image of size $W \times H$,

$$\#\text{retrieved templates} \;/\; \frac{\#\text{templates in database} \cdot W \cdot H}{T \cdot T} \tag{5.5}$$

stays small as shown in Figure 5.8 (right) and explains our improvement in comparison to an exhaustive search: while increasing the object database size, the ratio grows smaller. It is this trend of decay that allows us to scale sublinearly with the number of objects/views.

5.5 Conclusion

We presented a novel method for 3D object detection and pose estimation which employs hashing for efficient and truly scalable view-based matching against RGB-D frames. We showed that we outperform similar methods in terms of speed while being able to achieve comparable accuracies on a challenging dataset.

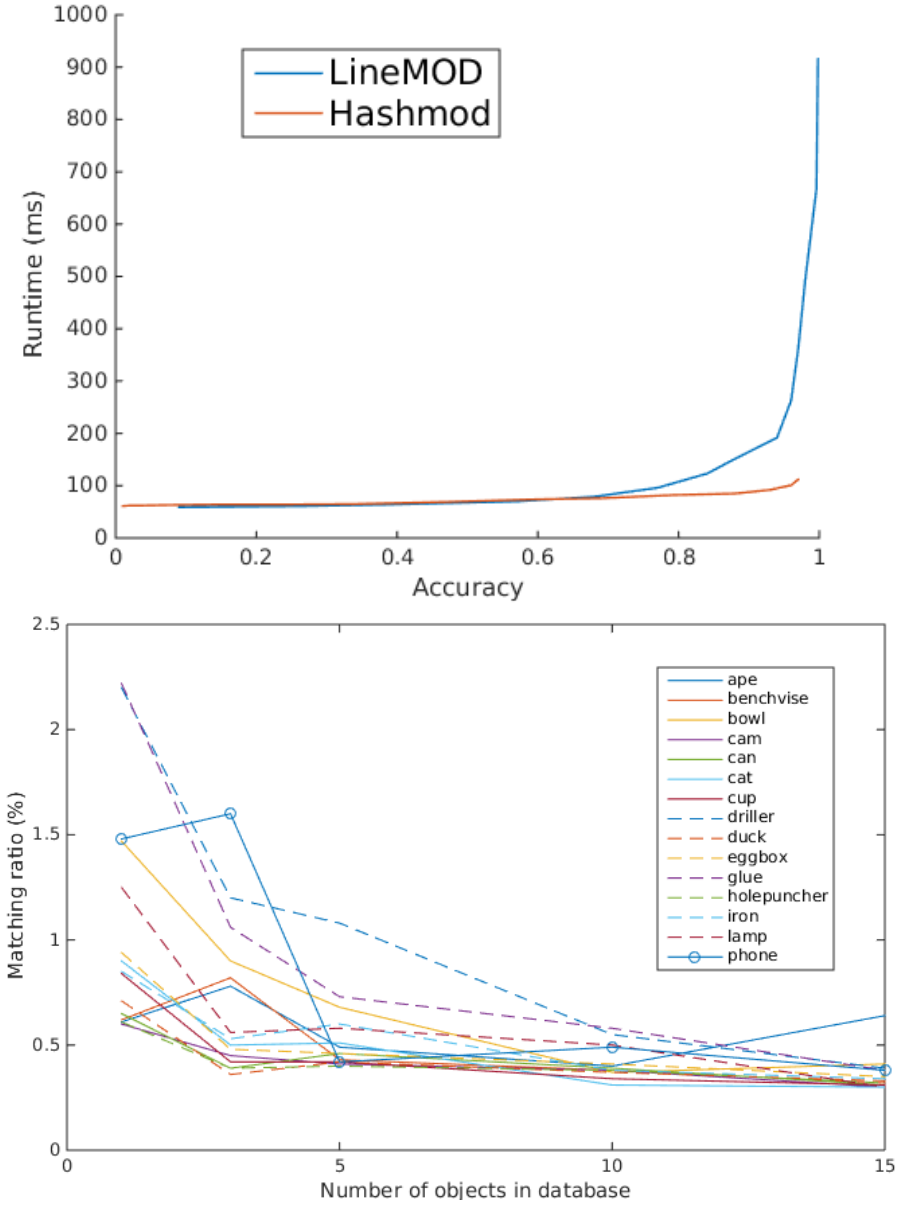


Figure 5.8: Top: Accuracy versus runtime on the 'driller'-sequence with TBV hash keys and LineMOD with a set of decreasing matching thresholds. Both methods achieve higher accuracies with a lower threshold although we only retrieve a fraction of views, making our runtime increase marginal. Bottom: Matching ratios for an increasing number of objects using TBV hash keys. The obvious decreasing trend allows us to scale with the number of objects.

Chapter 6

Convolutional Autoencoders for 6D Instance Voting

In this chapter we present a 3D object detection method that uses regressed descriptors of locally-sampled RGB-D patches for 6D vote casting. To this end, we deeply learn descriptive features and use them to create hypotheses in the 6D pose space, similar to [Brachmann et al., 2014, Tejani et al., 2014].

In practice, we train a convolutional autoencoder (CAE) [Masci et al., 2011] from scratch using random patches from RGB-D images with the goal of descriptor regression. With this network we create codebooks from synthetic patches sampled from object views. For detection, we sample patches in the input image on a regular grid, compute the patch descriptors and match them against codebooks with an approximate k-NN search. Additionally, we associate to each codebook patch a vote that corresponds to object orientation and centroid offset. Matching returns a number of candidate votes which are cast only if their

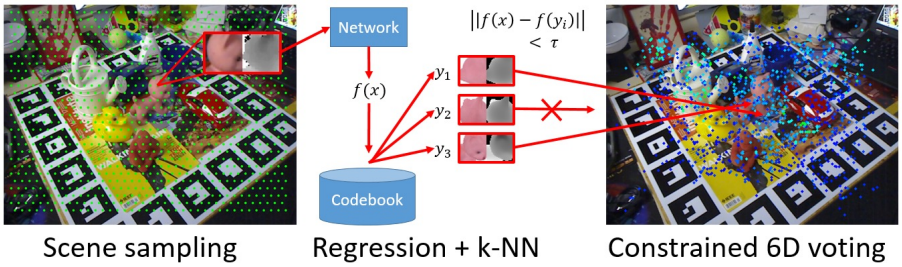


Figure 6.1: Illustration of the voting. The scene is densely sampled to extract scale-invariant RGB-D patches which are fed into a network to regress features for a subsequent k-NN search in a codebook of precomputed synthetic local object patches. The retrieved neighbors then cast 6D votes if their feature distance is smaller than a threshold τ.

matching score surpasses a threshold. In Figure 6.1 we show a schematic drawing of the detection pipeline. We will show that our method allows for training on real data, efficient matching between synthetic and real patches and that it generalizes well to unseen data with an extremely high recall. Furthermore, our approach avoids explicit background learning and scales well with the number of objects in the database.

We first give a description of how we sample local RGB-D patches of the given target objects and the scene while ensuring scale-invariance and suitability as a neural network input. Secondly, we describe the employed neural networks in more detail. Finally, we present our voting and filtering approach which efficiently detects objects in real scenes using a trained network and a codebook of regressed descriptors from synthetic patches.

6.1 Local Patch Representation

Our method follows an established paradigm for voting via local information. Given an object appearance, the idea is to separate it into its local parts and vote independently [Pepik et al., 2012, Gall et al., 2011, Tejani et al., 2014]. While most approaches rely on hand-crafted features for describing these local patches, we tackle the issue by regressing them with a neural network.

To represent an object locally, we render it from many viewpoints equidistantly sampled on an icosahedron, similar to [Hinterstoisser et al., 2012b]. The angular distance between two vertices on the icosahedron is approximately 8 degrees with our sampling. Then, a set of scale-independent RGB-D patches is densely extracted at each view. To sample invariantly to scale, we take depth Z at the patch center point and compute the patch size in pixels such that the patch would correspond to a fixed metric size m (here: 5 cm) via

$$patch_{size} = \frac{m}{Z} \cdot f \tag{6.1}$$

with f being the focal length of the rendering camera. After patch extraction, we de-mean the depth values with Z and clamp them to $\pm m$ to confine the patch locally not only along X and Y, but also along z. Finally, we normalize color and depth to $[-1, 1]$ and resize the patches to 32×32. See Figure 6.2 for an exemplary synthetic view together with sampled local patches.

An important advantage of using local patches as in the proposed framework is that it avoids the problematic aspect of background modeling. Indeed, for what concerns discriminative approaches based on learning a background and a foreground class, a generic background appearance can hardly be modeled, and recent approaches based on discriminative classifiers such as CNNs deploy scene data for training, thus becoming extremely dataset-specific and necessitating refinement strategies such as hard negative mining. Also, both [Brachmann et al., 2014] and [Wohlhart and Lepetit, 2015] model a supporting plane to achieve improved results, with the latter even introducing real images intertwined with synthetic renderings into the training to force the CNN to abstract from real background. Our method instead does not need to model the background at all.

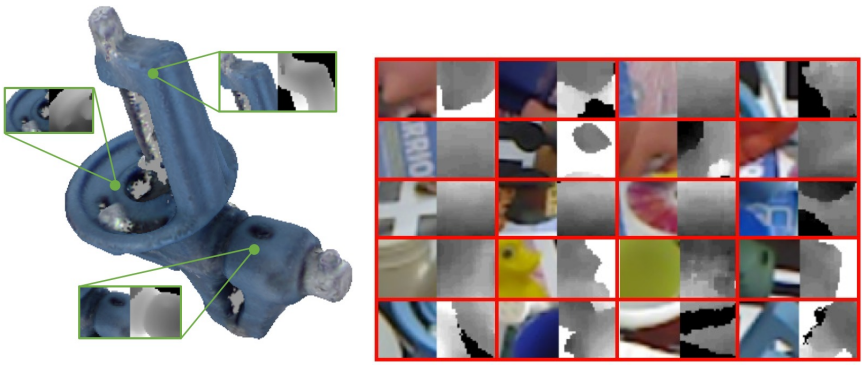


Figure 6.2: Left: For each synthetic view, we sample scale-invariant RGB-D patches $\mathbf{y}_i$ of a fixed metric size on a dense grid. Their associated regressed features $f(\mathbf{y}_i)$ and local votes $v(\mathbf{y}_i)$ are stored into a codebook. Right: Examples from the approx. 1.5 million random patches taken from the LineMOD dataset for autoencoder training.

6.2 Network Training

Since we want the network to produce discriminative features for the provided input RGB-D patches, we need to bootstrap suitable filters and weights for the intermediate layers of the network. In contrast to many other works that rely on pretrained, publicly available networks, we train from scratch due to multiple reasons:

1. Not many works have incorporated depth as an additional channel in networks and most remark that special care has to be taken to cope with, among others, sensor noise and depth 'holes' which we can control with our data.

2. We are one of the first to focus on local RGB-D patches of small-scale objects. There are no pretrained networks that have been so far learned on such data, and it is unclear how well other networks that were learned on RGB-D data can generalize to our specific problem at hand.

3. To robustly train deep architectures, a high amount of training samples is needed. By using patches from real scenes, we can easily create a huge training dataset which is specialized to our task, thus enhancing the discriminative power of our network.

Note that other works usually train a CNN on a classification problem and then use a 'beheaded' network for other tasks (e.g. [Girshick et al., 2014]). Here, we cannot simply convert our problem into a feasible classification task because of the sheer amount of training samples that range in the millions. Although we could assign each sample to the object class it belongs to, this would bias the feature training and hence, counter the learning of a generalized patch feature representation, independent of object affiliations. It is important to point out

that also [Wohlhart and Lepetit, 2015] aimed for feature learning, but with a different goal. Indeed, they enforce distance similarity of feature space and object pose space, while we instead strive for a compact representation of our local input patches, independent of the objects' poses.

We teach the network regression on a large variety of input data by randomly sampling local patches from the LineMOD dataset [Hinterstoisser et al., 2012b], amounting to around 1.5 million total samples. Furthermore, these samples were augmented such that each image got randomly flipped and its color channels permuted. Our network aims to learn a mapping from the high-dimensional patch space to a much lower feature space of dimensionality F, and we employ a traditional autoencoder (AE) and a convolutional autoencoder (CAE) to accomplish this task.

Autoencoders minimize a reconstruction error $||\mathbf{x} - \mathbf{y}||$ between an input patch $\mathbf{x}$ and a regressed output patch $\mathbf{y}$ while the inner-most compression layer condenses the data into F values. We use these F values as our descriptor since they represent the most informative compact encoding of the input patch. Our architectures can be seen in Figure 6.3. For the AE we use two encoding and decoding layers which are all connected with tanh activations. For the CAE we employ multiple layers of 5×5 convolutions and PReLUs (Parametrized Rectified Linear Unit) before a single fully-connected encoding/decoding layer, and use a deconvolution with learned 2×2 kernels for upscaling before proceeding back again with 5×5 convolutions and PReLUs. Note that we conduct one max-pool operation after the first convolutions to introduce a small shift-invariance.

6.3 Constrained Voting

A problem that is often encountered in regression tasks is the unpredictability of output values in the case of noisy or unseen, ill-conditioned input data. This is especially true for CNNs as a deep cascade of non-linear functions composed of many parameters. In our case, this can be caused by e.g., unseen object parts, general background appearance or sensor noise in color or depth. If we were to simply regress the translational and rotational parts, we would be prone to this input sensitivity. Furthermore, this approach would always cast votes at each image sampling position, increasing the complexity of sifting through the voting space afterwards. Instead, we render an object from many views and store local patches $\mathbf{y}$ of this synthetic data in a database, as seen in Figure 6.2. For each $\mathbf{y}$, we compute its feature $f(\mathbf{y}) \in \mathbb{R}^F$ and store it together with a local vote $(T_x, T_y, T_z, \alpha, \beta, \gamma)$ describing the patch 3D center point offset to the object centroid and the rotation with respect to the local object coordinate frame. This serves as an object-dependent codebook.

During testing, we take each sampled 3D scene point $\mathbf{S} = (S_x, S_y, S_z)$ with associated patch $\mathbf{x}$, compute its deep-regressed feature $f(\mathbf{x})$ and retrieve k (approximate) nearest neighbors $\mathbf{y}_1, ..., \mathbf{y}_k$. Each neighbor casts then a global vote $v(\mathbf{S}, \mathbf{y}) = (T_x + S_x, T_y + S_y, T_z + S_z, \alpha, \beta, \gamma)$ with an associated weight $w(v) = e^{-||f(\mathbf{x})-f(\mathbf{y})||}$ based on the feature distance.

Notably, this approach is flexible enough to provide three main practical advantages. First, we can vary k in order to steer the amount of possible vote

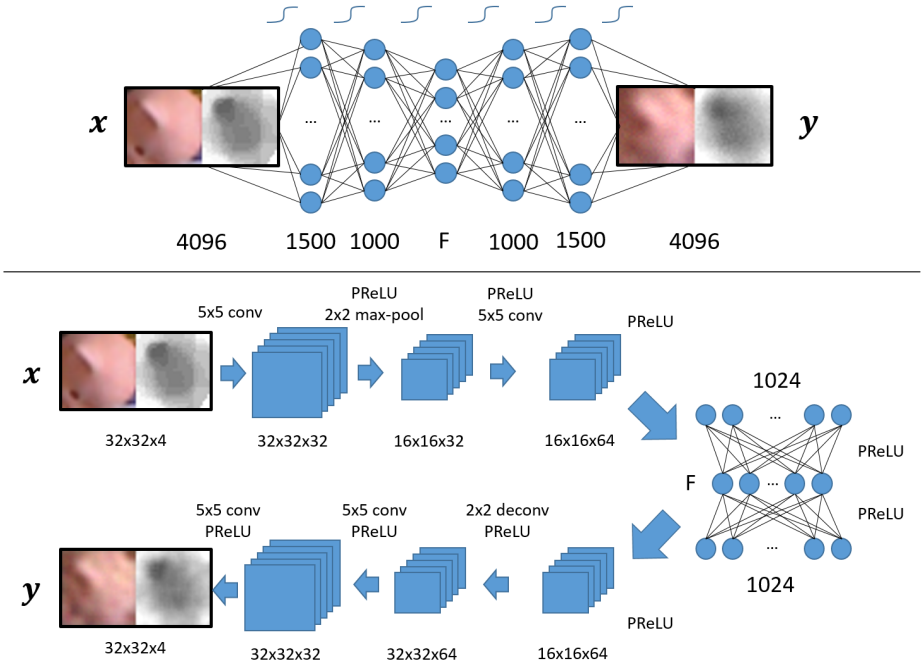


Figure 6.3: Depiction of the employed AE (top) and CAE (bottom) architectures. For both, we have the latent feature layer with dimensionality F.

candidates per sampling position. Together with a joint codebook for all objects, we can retrieve the nearest neighbors with sub-linear complexity, enabling scalability. Secondly, we can define a threshold τ on the nearest neighbor distance, so that retrieved neighbors will only vote if they hold a certain confidence. This reduces the amount of votes cast over scene parts that do not resemble any of the codebook patches. Furthermore, if noise sensitivity perturbs our regressed feature, it is more likely to be hindered from vote casting. Lastly and of significance, it is assured that each vote is numerically correct because it is unaffected by noise in the input data, given that the feature matching was reliable. See Figure 6.4 for a visualization of the constrained voting.

6.3.1 Vote filtering

Casting votes can lead to a very crowded vote space that requires refinement in order to keep detection computationally feasible. We thus employ a three-stage filtering: in the first stage we subdivide the image plane into a 2D grid (here: cell size of 5×5 pixels) and throw each vote into the cell the projected centroid points to. We suppress all cells that hold less than k votes and extract local maxima after bilinear filtering over the accumulated weights of the cells. Each local mode collects the votes from its direct cell neighbors and performs mean shift with a flat kernel, first in translational and then in quaternion space (here: kernel sizes 2.5 cm and 7 degrees). This filtering is computationally very efficient and removes most spurious votes with non-agreeing centroids, while retaining

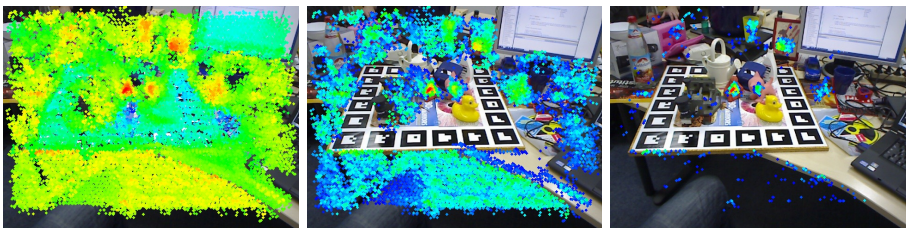


Figure 6.4: Casting the constrained votes for $k = 10$ with a varying distance threshold (left to right): $\tau = 15$, $\tau = 7$, $\tau = 5$. The projected vote centroids v_i are colored according to their scaled weight $w(v_i)/\tau$. It can be seen that many votes accumulate confidently around the true object centroid for differently chosen thresholds.

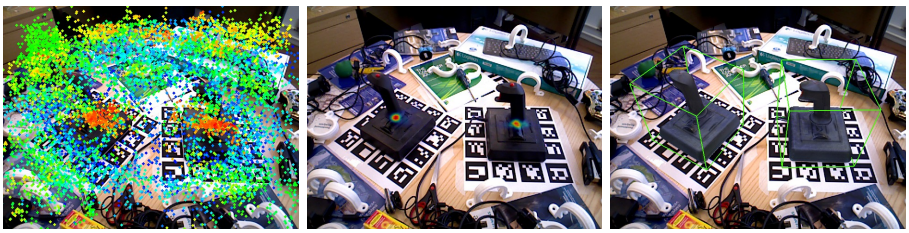


Figure 6.5: Starting with thousands of votes (left) we filter and retrieve intermediate local maxima (middle) that are further verified and accepted (right).

plausible hypotheses, as can be seen in Figure 6.5.

6.4 Evaluation

6.4.1 Reconstruction quality

To evaluate the performance of the networks, we trained AEs and CAEs with feature layer dimensions $F \in \{32, 64, 128, 256\}$. We implemented our networks with Caffe [Jia et al., 2014] and trained each with an NVIDIA Titan X with a batch size of 500. The learning rate was fixed to 10^{-5} and we ran 100,000 iterations for each network. The only exception was the 256-dim AE, which we trained for 200,000 iterations for convergence due to its higher number of parameters.

For a visual impression of the results, we present the reconstruction quality side-by-side of AEs and CAEs on six random RGB-D patches in Figure 6.6. Note that these patches are test samples from another dataset and thus have not been part of the training, i.e. the networks are reconstructing previously unseen data.

It is apparent that the CAEs put more emphasis on different image properties than their AE pendants. The AE reconstructions focus more on color and are more afflicted by noise since weights of neighboring pixels are trained in an uncorrelated fashion in this architecture. The CAE patches instead recover the spatial structure better at the cost of color fidelity. This can be especially seen

Figure 6.6: RGB-D patch reconstruction comparison between our AE and CAE for a given feature dimensionality F. Clearly, the AE and CAE focus on different qualities and both networks increase the reconstruction fidelity with a wider compression layer.

for the 64-dimensional CAE where the remaining $1.56\% = (64/4096)$ of the input information forced the network to resort to grayscale in order to preserve image structure. It can be objectively stated that the convolutional reconstructions for 128 dimensions are usually closer to their input in visual terms. Subsequently, at dimensionality 256 the CAE results are consistently of higher fidelity both in terms of structure and color/texture.

Feature retrieval quality For a visual feedback of the feature quality we refer to Figure 6.7. For each depicted object we took the first frame of the respective sequence and show the closest neighbor from the codebook ($\tau = \infty$). It is obvious that the features represent well the underlying visual appearance since the putative matches resemble each other well in color and depth.

6.4.2 Self-evaluation with changing parameters

Our method is mainly governed by three parameters: τ for constrained voting, k as the number of retrieved neighbors from the codebook, and the sampling density. We ran multiple experiments on the dataset of Tejani *et al.* and give further insight.

We first wanted to convey the importance of constrained voting via the left graph in Figure 6.8. Apparently, the threshold needs to reflect the dimensionality of the features, i.e. if the feature is of higher dimensionality, the norm difference $||f(\mathbf{x}) - f(\mathbf{y})||$ grows accordingly. Nonetheless, larger features are more descriptive: while initially both networks underperform since many correct votes are disallowed from being cast, CAE-64 reaches its peak performance already at around $\tau = 7$ and from there on additional votes add to more confusion and false positives in the scene. CAE-128 peaks at around $\tau = 10$ and

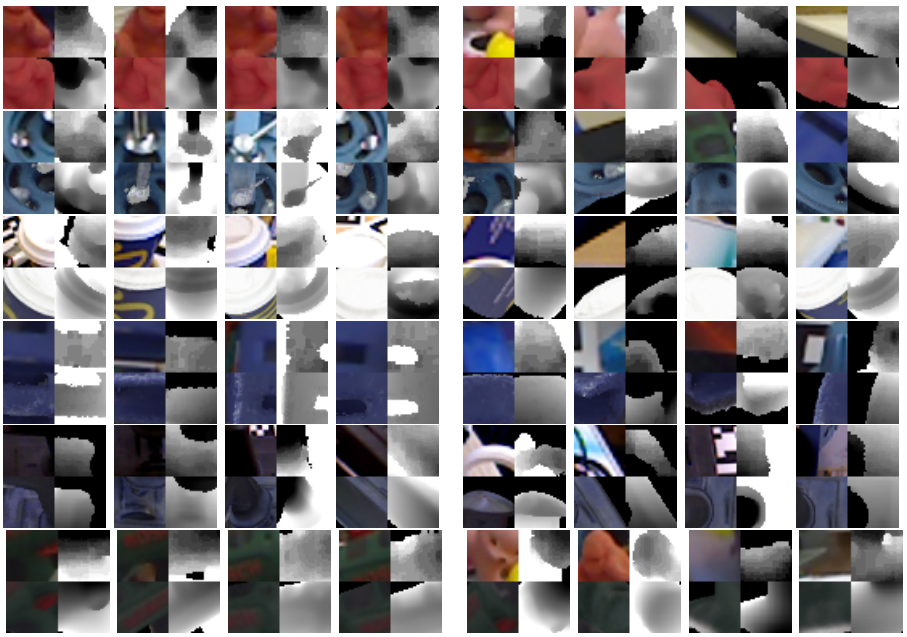


Figure 6.7: Putative RGB-D patch matches. For each scene input patch, we show the retrieved nearest neighbor from the synthetic model database. For easier distinction, we divided the matches up into correct (left column) and wrong (right column). As can be seen, the features do reflect the visual similarity quite well, even for wrong matches.

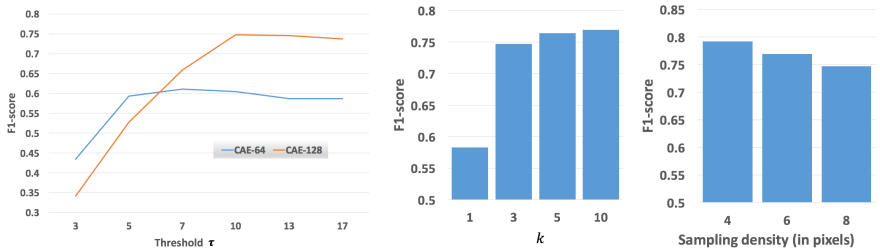


Figure 6.8: Evaluation of parameter influence. From left to right: threshold τ, number of retrieved neighbors k, sampling step size in pixels.

shows a similar behavior as CAE-64 for larger thresholds, albeit of smaller effect.

The number of retrieved neighbors k and the change in the F1-score can be seen in the center plot from Figure 6.8. Interestingly, the choice of k does not impact our general accuracy too much, apart from the initial jump from $k = 1$ to $k = 3$. This means that a good match between real and synthetic data is most often found among the first retrieved neighbors. Furthermore, our verification usually always decides correctly for the geometrically best fitting candidate if multiple hypotheses coincide at the same spatial position (centroids are closer than 5 cm). We also show in the right plot that a denser sampling improves the overall accuracy as expected. This was especially observable for the small "camera" as well as the "shampoo" that exhibits only its thin side at times and can be missed during sampling.

Unfortunately, with a higher k the runtime increases drastically since the number of hypotheses after mean shift can range in the hundreds per extracted mode. This is due to the fact that we cluster together all votes from the immediate neighbors for each local maximum. In turn, this can lead to multiple seconds of pose refinement and subsequent verification. The same happens with a finer sampling of the scene since the total number of scene votes has an upper bound of $\#samples \cdot k$, extremely cut down by τ in practice. We therefore fixed $k = 3$ and a sampling step of 8 as a reasonable compromise.

6.4.3 Multi-instance dataset from Tejani et al.

We evaluated our approach on the dataset of Tejani et al. [Tejani et al., 2014]. To evaluate against the authors' method (LC-HF), we follow their protocol and extract the $N = 5$ strongest modes in the voting space and subsequently verify each via ICP and depth/normal checks to suppress false positives.

We used this dataset first to evaluate how different networks and feature dimensions influence the final detection result. To this end, we fixed $k = 3$ and conducted a run with the trained CAEs, AEs and also compared to PCA[1] as means for dimensionality reduction. Since different dimensions and methods lead to different feature distances we set $\tau = \infty$ for this experiment, i.e. votes are unconstrained. Note that we already show here generalization since the networks were trained on patches from another dataset. As can be seen in Table

[1]Due to computational constraints we took only 1 million patches for PCA training.

F	32	64	128	256
PCA	0.33	0.43	0.46	0.47

F	32	64	128	256
AE	**0.43**	**0.63**	0.65	0.66

F	32	64	128	256
CAE	0.32	0.58	**0.70**	**0.69**

Table 6.1: F1-scores on the Tejani dataset using PCA, AE and CAE for patch descriptor regression with a varying dimension F. We highlight the best method for a given F. Also note that the number for CAE-128 deviates from Table 3 since we did not constrain the voting for this experiment.

6.1, PCA provides a good baseline performance that surpasses even the CAE at 32 dimensions, although this mainly stems from a high precision since vote centroids rarely agreed. In turn, both networks supplied similar behavior and we reached a peak at 128 with our CAE, which we fixed for further evaluation. We also found $\tau = 10$ and a sampling step of 8 pixels to provide a good balance between accuracy and runtime. For a more in-depth self comparison, we kindly refer the reader to the supplementary material.

For this evaluation we also supply numbers from the original implementation of LineMOD [Hinterstoisser et al., 2012a]. Since LineMOD is a matching-based approach, we evaluated such that each template having a similarity score larger than 0.8 is taken into the same verification described above. It is evident that LineMOD fares very well on most sequences since the amount of occlusion is low. It only showed problems where objects sometimes are partially outside the image plane (e.g. 'joystick','coffe'), have many occluders and thus a smaller recall ('milk') or where the planar 'juice' object decreased the precision by occasional misdetections in the table. Not surprisingly, LineMOD outperforms the other two methods largely for the small 'camera' since it searches the entire specified scale space whereas LC-HF and our method both rely on local depth for scaling. Although our local voting does detect instances in the table as well, there is rarely an agreeing centroid that survives the filtering stage and our method is by far more robust to larger occlusions and partial views. We are thus overtaking the other methods in 'coffe' and 'joystick'. The 'milk' object is difficult to handle with local methods since it is uniformly colored and symmetric, defying a reliable accumulation of vote centroids. Although the 'joystick' is mostly black, its geometry allows us to recover the pose very reliably. All in all, we outperform the state-of-the art in holistic matching slightly while clearly improving over the state-of-the-art in local-based detection by significant 9.6% on this challenging dataset. Detailed numbers are given in Tables 2 and 3. Unfortunately, runtimes for LC-HF are not provided by the authors.

6.4.4 ACCV12 dataset

We evaluated our method on the benchmark of [Hinterstoisser et al., 2012b] in two different ways. To compare to a whole set of related work that followed the original evaluation protocol, we remove the last stage of vote filtering and take the $N = 100$ most confident votes for the final hypotheses to decide for the

Figure 6.9: Scene sampling, vote maps and detection output for two objects on the Tejani dataset. Red sample points were skipped due to missing depth.

Stage	Runtime (ms)
Sampling	0.03
Regression	477.3
Voting	61.4
Filtering	1.6
Verification	130.5
Total	670.8

Table 6.2: Average runtime on [Tejani et al., 2014]. Note that feature regression runs on GPU.

Sequence	LineMOD	LC-HF	Us
Camera	**0.589**	0.394	0.383
Coffee	0.942	0.891	**0.972**
Joystick	0.846	0.549	**0.892**
Juice	0.595	**0.883**	0.866
Milk	**0.558**	0.397	0.463
Shampoo	**0.922**	0.792	0.910
Total	0.740	0.651	**0.747**

Table 6.3: F1-scores for each sequence on [Tejani et al., 2014].

	ape	bvise	bowl	cam	can	cat	cup	driller	duck
Us	**96.9**	94.1	**99.9**	97.7	95.2	97.4	**99.6**	96.2	**97.3**
LineMOD	95.8	98.7	**99.9**	97.5	95.4	**99.3**	97.1	93.6	95.9
Kehl2015	96.1	92.8	99.3	97.8	92.8	98.9	96.2	**98.2**	94.1
Rios-Cabrera2013	95.0	98.9	99.7	**98.2**	**96.3**	99.1	97.5	94.3	94.2
Hodan2015	93.9	**99.8**	98.8	95.5	95.9	98.2	99.5	94.1	94.3

	eggb	glue	holep	iron	lamp	phone
Us	99.9	78.6	96.8	**98.7**	96.2	92.8
LineMOD	99.8	91.8	95.9	97.5	97.7	93.3
Kehl2015	99.9	96.8	95.7	96.5	98.4	93.3
Rios-Cabrera2013	99.8	96.3	**97.5**	98.4	97.9	**95.3**
Hodan2015	**100**	**98.0**	88.0	97.0	88.8	89.4

Table 6.4: ADD scores for each sequence of [Hinterstoisser et al., 2012b].

best hypothesis and use the factor $k_m = 0.1$ in their proposed error measure. To evaluate against Tejani et al. we instead follow their protocol and extract the $N = 5$ strongest modes in the voting space and choose $k_m = 0.15$. Since the dataset provides one object ground truth per sequence, we use only the codebook that is associated to that object for retrieving the nearest neighbors. Two objects, namely 'cup' and 'bowl', are missing their meshed models which we manually created. For either protocol we eventually verify each hypothesis via a fast projective ICP followed by a depth and normal check. Results are given in Tables 6.4 and 6.5.

We compute mean precision over 13 objects used in [Brachmann et al., 2014] and report 95.2%. We are thus between their plane-trained model with an average of 98.3% and their noise-trained model of 92.6% on pure synthetic data. We fare relatively well with our detections and can position ourselves nicely between the other state-of-the-art approaches. We could observe that we have a near-perfect recall for each object and that our network regresses reliable features allowing to match between synthetic and real local patches. We regard this to be the most important finding of our work since achieving high precision on a dataset can be usually fine-tuned. Nonetheless, the recall for the 'glue' is rather low since it is thin and thus occasionally missed by our sampling. Based on the overall observation, our comparison of the F1-scores with [Tejani et al., 2014] gives further proof of the soundness of our method. We can present excellent numbers and also show some qualitative results of the votes and detections in Figure 6.10.

6.4.5 Challenge dataset

Lastly, we also evaluated on the 'Challenge' dataset used in [Aldoma et al., 2015] containing 35 objects in 39 tabletop sequences with varying amounts of occlusion. The related work usually combines many different cues and descriptors together with elaborate verification schemes to achieve their results. We use this dataset to convey three aspects: we can reliably detect multiple objects undergoing many levels of occlusion while attaining acceptable detection results, we show again generalization on unseen data and that we accomplish this at low runtimes. We present a comparison of our method and related methods in

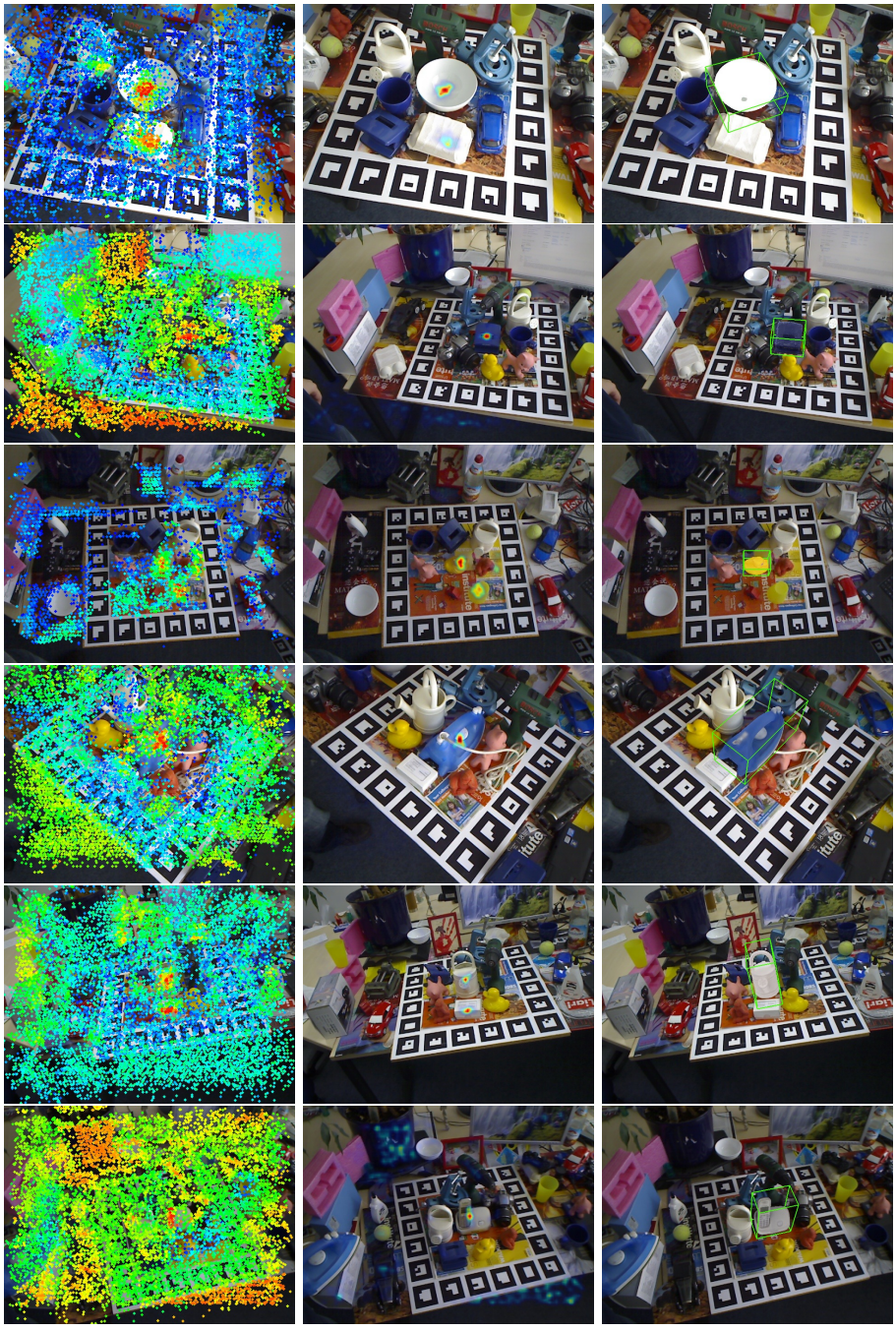


Figure 6.10: Showing vote maps, probability maps after filtering and detection output on some frames for different objects on the LineMOD dataset.

	ape	bvise	bowl	cam	can	cat	cup	driller	duck
Us	**98.1**	94.8	100	**93.4**	**82.6**	**98.1**	99.9	**96.5**	**97.9**
LineMOD	53.3	84.6	-	64.0	51.2	65.6	-	69.1	58.0
Tejani2014	85.5	**96.1**	-	71.8	70.9	88.8	-	90.5	90.7

	eggb	glue	holep	iron	lamp	phone
Us	**100**	**74.1**	**97.9**	**91.0**	**98.2**	**84.9**
LineMOD	86.0	43.8	51.6	68.3	67.5	56.3
Tejani2014	74.0	67.8	87.5	73.5	92.1	72.8

Table 6.5: F1-scores for each sequence of [Hinterstoisser et al., 2012b]. Note that these LineMOD scores are supplied from Tejani *et al.* with their evaluation since the original authors do not provide them. It is evident that our method performs by far better than the two competitors.

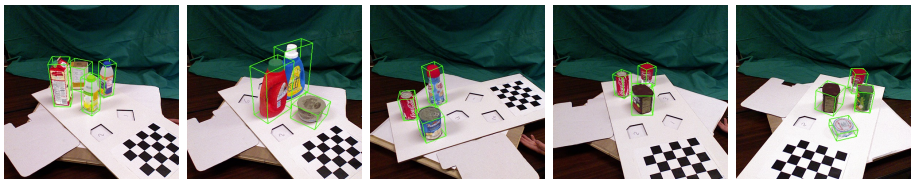


Figure 6.11: Detection output on selected frames from the 'Challenge' dataset.

Table 6 together with the average runtime per frame in Figure 11. Since we do not employ a computationally heavy verification the precision of our method is the lowest due to false positives surviving the checks. Nonetheless, we have a surprisingly high recall with our feature regression and voting scheme that brings our F1-score into a favorable position. It is important to note here that the related works employ a combination of local and global shape descriptors often directly processing the 3D point cloud, exploiting different color, texture and geometrical cues and this taking up to 10-20 seconds per frame. Instead, although our method does not attain such accuracy, it still provides higher efficiency thanks to the use of RGB-D patches only, as well as good scalability with the number of objects due to our discrete sampling (leading to an upper bound on the number of retrieved candidates) and approximate nearest-neighbor retrieval relying on sub-linear methods.

6.5 Conclusion

We showed that convolutional auto-encoders have the ability to regress meaningful and discriminative features from local RGB-D patches even for previously unseen input data, facilitating our method and allowing for robust multi-object and multi-instance detection under various levels of occlusion. Furthermore, our vote casting is inherently scalable and the introduced filtering stage allows to suppress many spurious votes. One main observation is that CAEs can abstract enough to reliably match between real and synthetic data. It is still unclear how a more refined training can further increase the results since different ar-

Method	Prec.	Recall	F1
GHV	1.0	0.99	0.99
Tang2011	0.98	0.90	0.94
Xie2013	1.0	0.99	0.99
Aldoma2013	0.99	0.99	0.99
Us	0.94	0.97	0.95

Table 6.6: Results on the 'Challenge' dataset. Although slightly weaker, our approach is far simpler and faster than related work.

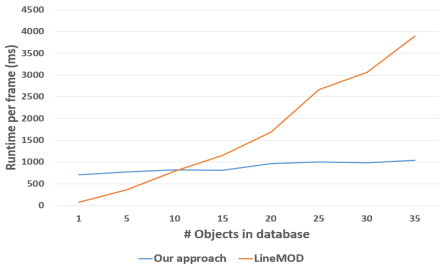


Figure 6.12: Average runtime on 'Challenge' with a changing amount of objects in the database.

chitectures have a tremendous impact on the network's performance.

Another problematic aspect is the complexity of hypothesis verification which can increase exponentially with the amount of objects and votes. Having a method that can combine local and holistic voting together with a learned verification into a single framework could lead to a higher detection precision.

Chapter 7

Single-Shot Detection for 6D Pose Estimation

The goal of this chapter is to present a deep network for object detection that can accurately deal with 3D models and 6D pose estimation by assuming an RGB image as unique input at test time. To this end, we bring the concept of SSD [Liu et al., 2016] over to this domain with the following contributions:

- a training stage that makes use of synthetic 3D model information only
- a decomposition of the model pose space that allows for easy training and handling of symmetries
- an extension of SSD that produces 2D detections and infers proper 6D poses

We argue that in most cases, color information alone can already provide close to perfect detection rates with good poses. Although our method does not need depth data, it is readily available with RGB-D sensors and almost all recent state-of-the-art 3D detectors make use of it for both feature computation and final pose refinement. We will thus treat depth as an optional modality for hypothesis verification and pose refinement and will assess the performance of our method with both 2D and 3D error metrics on multiple challenging datasets for the case of RGB and RGB-D data.

Throughout experimental results on multiple benchmark datasets, we demonstrate that our color-based approach is competitive with respect to state-of-the-art detectors that leverage RGB-D data (e.g. the method from our last chapter) or can even outperform them, while being many times faster. Indeed, we show that the prevalent trend of overly relying on depth for 3D instance detection is not justified when using color correctly.

The input to our method is an RGB image that is processed by the network to output localized 2D detections with bounding boxes. Additionally, each 2D box is provided with a pool of the most likely 6D poses for that instance. To represent a 6D pose, we parse the scores for viewpoint and in-plane rotation that have been inferred from the network and use projective properties to instantiate

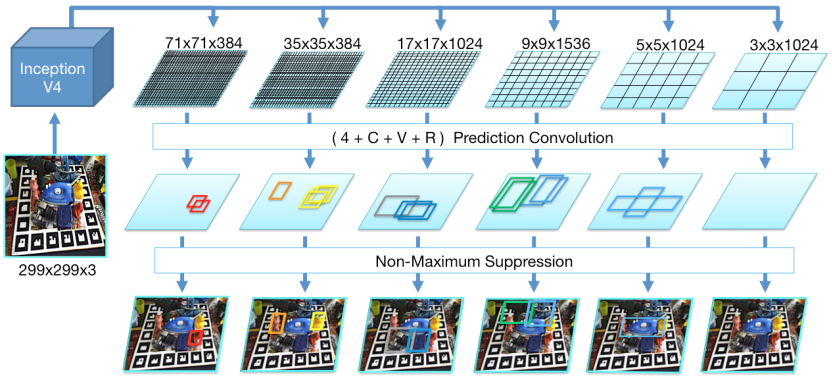


Figure 7.1: Schematic overview of the SSD-style network prediction. We feed our network with a 299 × 299 RGB image and produce six feature maps at different scales from the input image using branches from InceptionV4. Each map is then convolved with trained prediction kernels of shape (4 + C + V + R) to determine object class, 2D bounding box as well as scores for possible viewpoints and in-plane rotations that are parsed to build 6D pose hypotheses. Thereby, C denotes the number of object classes, V the number of viewpoints and R the number of in-plane rotation classes. The other 4 values refine the corners of the discrete bounding boxes to tightly fit the detected object.

6D hypotheses. In a final step, we refine each pose in every pool and select the best after verification. This last step can either be conducted in 2D or optionally in 3D if depth data is available. We present each part now in more detail.

7.1 Network architecture

Different to the original SSD work that uses a VGG16 backbone, we found that a pretrained InceptionV4 network [Szegedy et al., 2016] works better for our purposes and we feed with a color image (resized to 299 × 299) to compute feature maps at multiple scales.

Specifically, each of these six feature maps is convolved with prediction kernels that are supposed to regress localized detections from feature map positions. Let (w_s, h_s, c_s) be the width, height and channel depth at scale s. For each scale, we train a $3 \times 3 \times c_s$ kernel that provides for each feature map location the scores for object ID, discrete viewpoint and in-plane rotation. Since we introduce a discretization error by this grid, we create B_s bounding boxes at each location with different aspect ratios. Additionally, we regress a refinement of their four corners. If C, V, R are the numbers of object classes, sampled viewpoints and in-plane rotations respectively, we produce a $(w_s, h_s, B_s \times (C+V+R+4))$ detection map for the scale s. The network has a total number of 21222 possible bounding boxes in different shapes and sizes. While this might seem high, the actual runtime of our method is remarkably low thanks to the fully-convolutional design and the good true negative behavior, which tend to yield a very confident and small set of detections. We refer to Figure 7.1 for a schematic overview.

7.1.1 Viewpoint scoring versus pose regression

The choice of viewpoint classification over pose regression is deliberate. Direct rotation regression [Kendall et al., 2015, Tan et al., 2015] is possible but early experimentation showed clearly that the classification approach is more reliable for the task of detecting poses. In particular, it seems that the layers do a better job at scoring discrete viewpoints than at outputting numerically accurate translations and rotations. The decomposition of a 6D pose in viewpoint and in-plane rotation is elegant and allows us to tackle the problem more naturally. While a new viewpoint exhibits a new visual structure, an in-plane rotated view is a non-linear transformation of the same view. Furthermore, simultaneous scoring of all views allows us to parse multiple detections at a given image location, e.g. by accepting all viewpoints above a certain threshold. Equally important, this approach allows us to deal with symmetries or views of similar appearance in a straight-forward fashion.

7.2 Training stage

We take random images from MS COCO [Lin et al., 2014] as background and render our objects with random transformations into the scene using OpenGL commands. For each rendered instance, we compute the IoU (intersection over union) of each box with the rendered mask and every box b with IoU > 0.5 is taken as a positive sample for this object class. Additionally, we determine for the used transformation its closest sampled discrete viewpoint and in-plane rotation as well as set its four corner values to the tightest fit around the mask as a regression target. We show some training images in Figure 7.2.

Similar to SSD [Liu et al., 2016], we employ many different kinds of augmentation, such as changing the brightness and contrast of the image. Differently to them, though, we do not flip the images since it would lead to confusion between views and to wrong pose detections later on. We also make sure that each training image contains a 1:2 positives-negatives ratio by selecting hard negatives (unassigned boxes with high object probability) during back-propagation.

Our loss resembles the MultiBox loss of SSD or YOLO [Redmon et al., 2016], but we extend the formulation to take discrete views and in-plane rotations into account. Given a set of positive boxes Pos and hard-mined negative boxes Neg for a training image, we minimize the following energy:

$$L(Pos, Neg) := \sum_{b \in Neg} L_{class} + \sum_{b \in Pos} (L_{class} + \alpha L_{fit} + \beta L_{view} + \gamma L_{inplane}) \tag{7.1}$$

As it can be seen from (7.1), we sum over positive and negative boxes for class probabilities (L_{class}). Additionally, each positive box contributes weighted terms for viewpoint (L_{view}) and in-plane classification ($L_{inplane}$), as well as a fitting error of the boxes' corners (L_{fit}). For the classification terms, i.e., L_{class}, L_{view}, $L_{inplane}$, we employ a standard softmax cross-entropy loss, whereas a more robust smooth L1-norm is used for corner regression (L_{fit}).

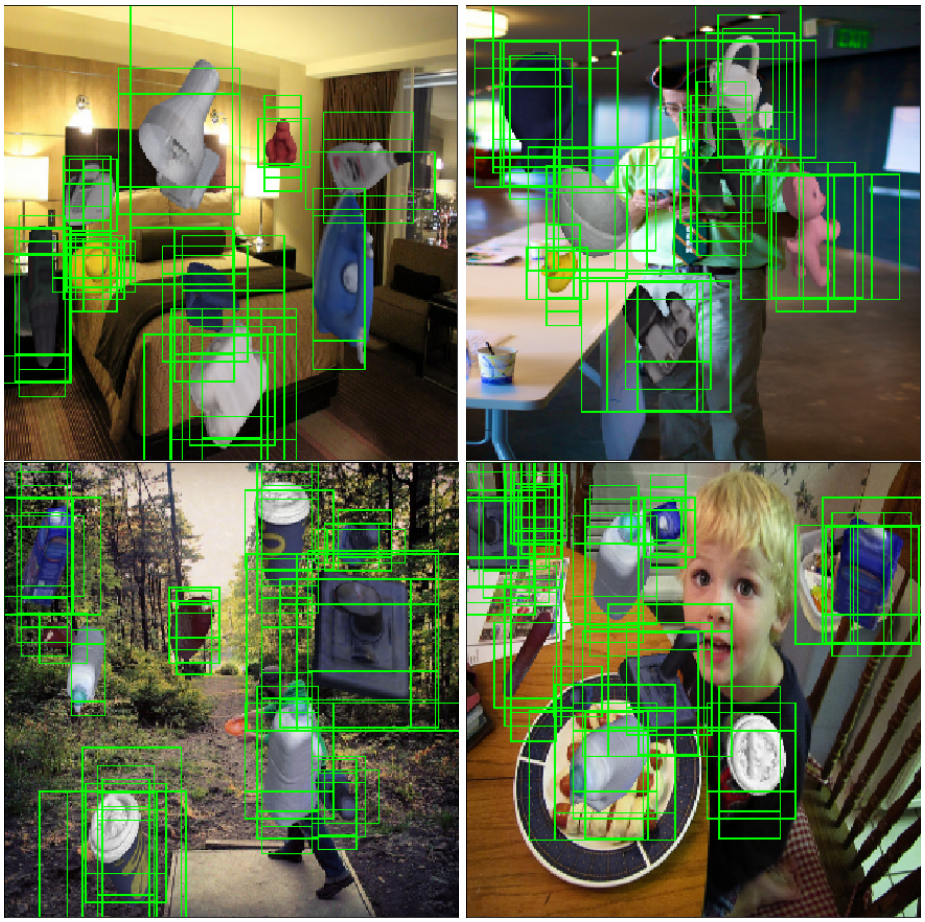


Figure 7.2: Exemplary training images for the datasets used. Using MS COCO images as background, we render object instances with random poses into the scene. The green boxes visualize the network's bounding boxes that have been assigned as positive samples for training.

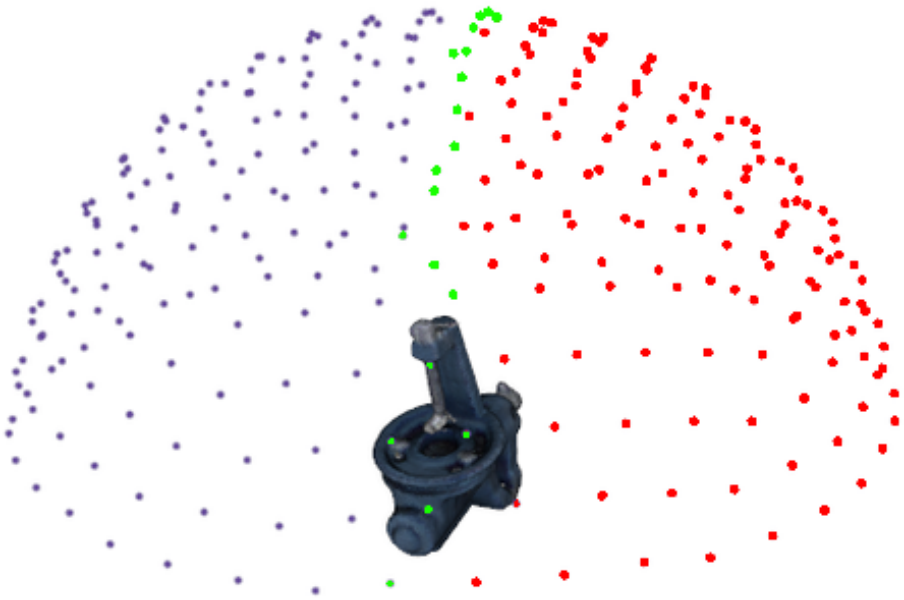


Figure 7.3: Discrete 6D pose space with each point representing a classifiable viewpoint. If symmetric, we use only the green points for view ID assignment during training whereas semi-symmetric objects use the red points as well.

7.2.1 Dealing with symmetry and view ambiguity

Our approach demands the elimination of viewpoint confusion for proper convergence. We thus have to treat symmetrical or semi-symmetrical (constructible with plane reflection) objects with special care. Given an equidistantly-sampled sphere from which we take our viewpoints, we discard positions that lead to ambiguity. For symmetric objects, we solely sample views along an arc, whereas for semi-symmetric objects we omit one hemisphere entirely. This approach easily generalizes to cope with views which are mutually indistinguishable although this might require manual annotation for specific objects in practice. In essence, we simply ignore certain views from the output of the convolutional classifiers during testing and take special care of viewpoint assignment in training. We refer to Figure 7.3 for a visualization of the pose space.

7.3 Detection stage

We run a forward-pass on the input image to collect all detections above a certain threshold, followed by non-maximum suppression. This yields refined and tight 2D bounding boxes with an associated object ID and scores for all views and in-plane rotations. For each detected 2D box we thus parse the most confident views as well as in-plane rotations to build a pool of 6D hypotheses from which we select the best after refinement. See Figure 7.5 for the pooled hypotheses and Figure 7.6 for the final output.

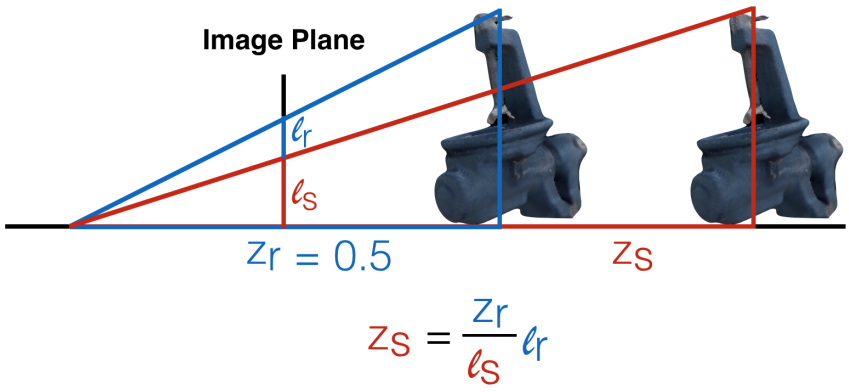


Figure 7.4: For each object we precomputed the perfect bounding box and the 2D object centroid with respect to each possible discrete rotation in a prior offline stage. To this end, we rendered the object at a canonical centroid distance $z_r = 0.5m$. Subsequently, the object distance z_s can be inferred from the projective ratio according to $z_s = \frac{l_r}{l_s} z_r$, where l_r denotes diagonal length of the precomputed bounding box and l_s denotes the diagonal length of the predicted bounding box on the image plane.

7.3.1 From 2D bounding box to 6D hypothesis

So far, all computation has been conducted on the image plane and we need to find a way to hypothesize 6D poses from our network output. We can easily construct a 3D rotation, given view ID and in-plane rotation ID, and can use the bounding box to infer 3D translation. To this end, we render all possible combinations of discrete views and in-plane rotations at a canonical centroid distance $z_r = 0.5m$ in an offline stage and compute their bounding boxes. Given the diagonal length l_r of the bounding box during this offline stage and the one predicted by the network l_r, we can infer the object distance $z_s = \frac{l_r}{l_s} z_r$ from their projective ratio, as illustrated in Figure 7.4. In a similar fashion, we can derive the projected centroid position and back-project to a 3D point with known camera intrinsics.

7.3.2 Pose refinement and verification

The obtained poses are already quite accurate, yet can in general benefit from a further refinement. Since we will regard the problem for both RGB and RGB-D data, the pose refinement will either be done with an edge-based or cloud-based ICP approach. If using RGB only, we render each hypothesis into the scene and extract a sparse set of 3D contour points. Each 3D point $\mathbf{X}_i$, projected to $\pi(\mathbf{X}_i) = \mathbf{x}_i$, then shoots a ray perpendicular to its orientation to find the closest scene edge $\mathbf{y}_i$. We seek the best alignment of the 3D model such that

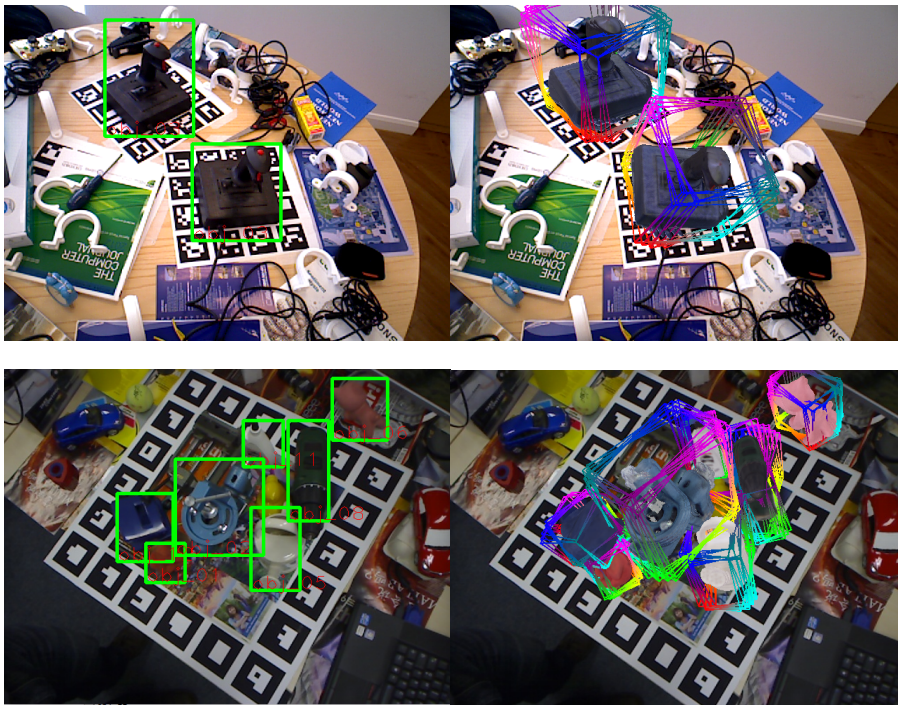


Figure 7.5: Prediction output and 6D pose pooling of our network on the Tejani dataset and the multi-object dataset. Each 2D prediction builds a pool of 6D poses by parsing the most confident views and in-plane rotations. Since our networks are trained with various augmentations, they can adapt to different global illumination settings.

the average projected error is minimal:

$$\arg\min_{\mathbf{R},\mathbf{T}} \sum_i \left(||\pi(\mathbf{R} \cdot \mathbf{X}_i + \mathbf{T}) - \mathbf{y}_i||^2 \right). \tag{7.2}$$

We minimize this energy with an IRLS approach and robustify it using Geman-McLure weighting (similar to [Drummond and Cipolla, 2002]). In the case of RGB-D, we render the current pose and solve with standard projective ICP with a point-to-plane formulation in closed form [Besl and McKay, 1992]. In both cases, we run multiple rounds of correspondence search to improve refinement and we use multi-threading to accelerate the process.

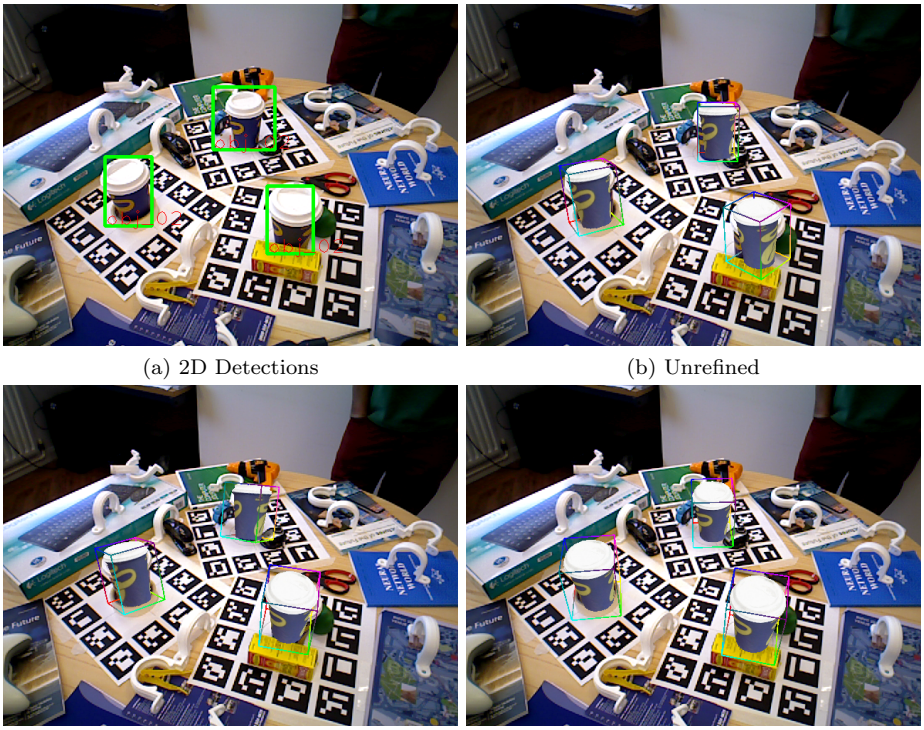
The above procedure provides multiple refined poses for each 2D box and we need to choose the best one. To this end, we employ a verification procedure. Using only RGB, we do a final rendering and compute the average deviation of orientation between contour gradients and overlapping scene gradients via absolute dot products. In case RGB-D data is available, we render the hypotheses and estimate camera-space normals to measure the similarity again with absolute dot products.

7.4 Evaluation

We implemented our method in C++ using TensorFlow [Abadi et al., 2016] ran it on a i7-5820K@3.3GHz with an NVIDIA GTX 1080. Our evaluation has been conducted on three datasets. The first two datasets are the same from the previous chapter, namely Tejani [Tejani et al., 2014] and ACCV12 [Hinterstoisser et al., 2012b]. The third, presented in [Brachmann et al., 2014], is an extension of the second where one sequence has been annotated with instances of multiple objects undergoing heavy occlusions at times.

Network configuration and training To get the best results it is necessary to find an appropriate sampling of the model view space. If the sampling is too coarse we either miss an object in certain poses or build suboptimal 6D hypotheses whereas a very fine sampling can lead to a more difficult training. We found an equidistant sampling of the unit sphere into 642 views to work well in practice. Since the datasets only exhibit the upper hemisphere of the objects, we ended up with 337 possible view IDs. Additionally, we sampled the in-plane rotations from -45 to 45 degrees in steps of 5 to have a total of 19 bins.

Given the above configuration, we trained the last layers of the network and the predictor kernels using ADAM and a constant learning rate of 0.0003 until we saw convergence on a synthetic validation set. The balancing of the loss term weights proved to be vital to provide both good detections and poses. After multiple trials we determined $\alpha = 1.5$, $\beta = 2.5$ and $\gamma = 1.5$ to work well for us. We refer the reader to the supplementary material to see the error development for different configurations.



(a) 2D Detections (b) Unrefined

(c) RGB refinement (d) RGB-D refinement

Figure 7.6: After predicting 2D detections (a), we build 6D hypotheses and run pose refinement and a final verification. While the unrefined poses (b) are rather approximate, contour-based refinement (c) produces already visually acceptable results. Occlusion-aware projective ICP with cloud data (d) leads to a very accurate alignment.

Sequence	LineMOD	LC-HF	Kehl2016a	Us
Camera	0.589	0.394	0.383	**0.741**
Coffee	0.942	0.891	0.972	**0.983**
Joystick	0.846	0.549	0.892	**0.997**
Juice	0.595	0.883	0.866	**0.919**
Milk	0.558	0.397	0.463	**0.780**
Shampoo	**0.922**	0.792	0.910	0.892
Total	0.740	0.651	0.747	**0.885**

Table 7.1: F1-scores on the re-annotated version of [Tejani et al., 2014]. Although our method is the only one to solely use RGB data, our results are considerably higher than all related work.

	ape	bvise	cam	can	cat	driller	duck	box
Our method	76.3	**97.1**	92.2	**93.1**	89.3	**97.8**	80.0	93.6
Kehl2016	**98.1**	94.8	**93.4**	82.6	**98.1**	96.5	**97.9**	**100**
LineMOD	53.3	84.6	64.0	51.2	65.6	69.1	58.0	86.0
LC-HF	85.5	96.1	71.8	70.9	88.8	90.5	90.7	74.0

	glue	holep	iron	lamp	phone
Our method	**76.3**	71.6	**98.2**	93.0	**92.4**
Kehl2016	74.1	**97.9**	91.0	**98.2**	84.9
LineMOD	43.8	51.6	68.3	67.5	56.3
LC-HF	67.8	87.5	73.5	92.1	72.8

Table 7.2: F1-scores for each sequence of [Hinterstoisser et al., 2012b]. Note that the LineMOD scores are supplied from [Tejani et al., 2014] with their evaluation since [Hinterstoisser et al., 2012b] does not provide them. Using color only we can easily compete with the other RGB-D based methods.

7.4.1 Single object scenario

Since 3D detection is a multi-stage pipeline for us, we first evaluate purely the 2D detection performance between our predicted boxes and the tight bounding boxes of the rendered groundtruth instances on the first two datasets. Note that we always conduct proper detection and not localization, i.e. we do not constrain the maximum number of allowed detections but instead accept all predictions above a chosen threshold. We count a detection to be correct when the IoU score of a predicted bounding box with the groundtruth box is higher than 0.5. We present our F1-scores in Tables 7.1 and 7.2 for different detection thresholds.

It is important to mention that the compared methods, which all use RGB-D data, allow a detection to survive after rigorous color- and depth-based checks whereas we use simple thresholding for each prediction. Therefore, it is easier for them to suppress false positives to increase their precision whereas our confidence comes from color cues only.

On the Tejani dataset we outperform all related RGB-D methods by a huge margin of 13.8% while using color only. We analyzed the detection quality on the two most difficult sequences. The 'camera' has instances of smaller scale which are partially occluded and therefore simply missed whereas the 'milk' sequence exhibits stronger occlusions in virtually every frame. Although we were able to detect the 'milk' instances, our predictors could not overcome the occlusions and regressed wrongly-sized boxes which were not tight enough to satisfy the IoU threshold. These were counted as false positives and thus lowered our recall. We refer to the appendix for more detailed graphs.

On the second dataset we have mixed results where we can outperform state-of-the-art RGB-D methods on some sequences while being worse on others. For larger feature-rich objects like 'benchvise', 'iron' or 'driller' our method performs better than the related work since our network can draw from color and textural information. For some objects, such as 'lamp' or 'cam', the performance is

Sequence	IoU-2D	IoU-3D	VSS-2D	VSS-3D
Camera	0.973	0.904	0.693	0.778
Coffee	0.998	0.996	0.765	0.931
Joystick	1	0.953	0.655	0.866
Juice	0.994	0.962	0.742	0.865
Milk	0.970	0.990	0.722	0.810
Shampoo	0.993	0.974	0.767	0.874
Total	0.988	0.963	0.724	0.854

Table 7.3: Average pose errors for the Tejani dataset.

	RGB		
	Ours	Brachmann2016	LineMOD
IoU	99.4 %	97.5%	86.5%
ADD	76.3%	50.2%	24.2%

worse than the related work. Our method relies on color information only and thus requires a certain color similarity between synthetic renderings of the CAD model and their appearance in the scene. Some objects exhibit specular effects (i.e. changing colors for different camera positions) or the frames can undergo sensor-side changes of exposure or white balancing, causing a color shift. Brachmann *et al.* [Brachmann et al., 2016] avoid this problem by training on a well-distributed subset of real sequence images. Our problem is much harder since we train on synthetic data only and must generalize to real, unseen imagery.

Our performance for objects of smaller scale such as 'ape', 'duck' and 'cat' is worse and we observed a drop both in recall and precision. We attribute the lower recall to our bounding box placement, which can have 'blind spots' at some locations and consequently, leading to situations where a small-scale instance cannot be covered sufficiently by any box to fire. The lower precision, on the other hand, stems from the fact that these objects are textureless and of uniform color which increases confusion with the heavy scene clutter.

Pose estimation

We chose for each object the threshold that yielded the highest F1-score and run all following pose estimation experiments with this setting. We are interested in the pose accuracy for all correctly detected instances.

	RGB-D		
	Ours	Brachmann2016	Brachmann2014
IoU	96.5 %	99.6%	99.1%
ADD	90.9%	99.0%	97.4%

Table 7.4: Average pose errors for the LineMOD dataset.

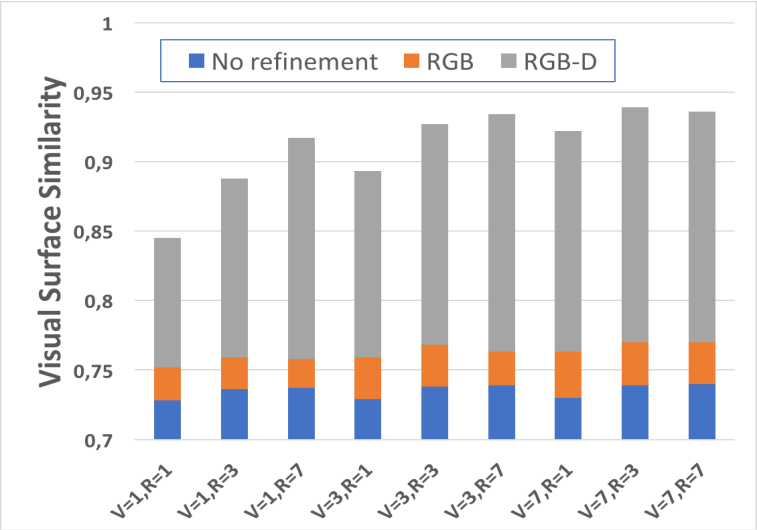


Figure 7.7: Average VSS scores for the 'coffee' object for different numbers of parsed views and in-plane rotations as well as different pose refinement options.

Error metrics To measure 2D pose errors we will compute both an IoU score and a Visual Surface Similarity (VSS) [Hodan et al., 2016]. The former is different than the detection IoU check since it measures the overlap of the rendered masks' bounding boxes between groundtruth and final pose estimate and accepts a pose if the overlap is larger than 0.5. VSS is a tighter measure since it counts the average pixel-wise overlap of the mask. This measure assesses well the suitability for AR applications and has the advantage of being agnostic towards the symmetry of objects. To measure the 3D pose error we use the ADD score from [Hinterstoisser et al., 2012b]. This assesses the accuracy for manipulation tasks by measuring the average deviation between transformed model point clouds of groundtruth and hypothesis. If it is smaller than $\frac{1}{10}th$ of the model diameter, it is counted as a correct pose.

Refinement with different parsing values As mentioned, we parse the most confident views and in-plane rotations to build a pool of 6D hypotheses for each 2D detection. Here, we want to assess the final pose accuracy when changing the number of parsed views V and rotations R for different refinement strategies We present in Figure 7.7 the results on Tejani's 'coffee' sequence for the cases of no refinement, edge-based and cloud-based refinement (see Figure 7.6 for an example). To decide for the best pose we employ verification over contours for the first two cases and normals for the latter. As can be seen, the final poses without any refinement are imperfect but usually provide very good initializations for further processing. Additional 2D refinement yields better poses but cannot cope well with occluders whereas depth-based refinement leads to perfect poses in practice. The figure gives also insight for varying V and R for hypothesis pool creation. Naturally, with higher numbers the chances of finding a more accurate pose improve since we evaluate a larger portion of the 6D space.

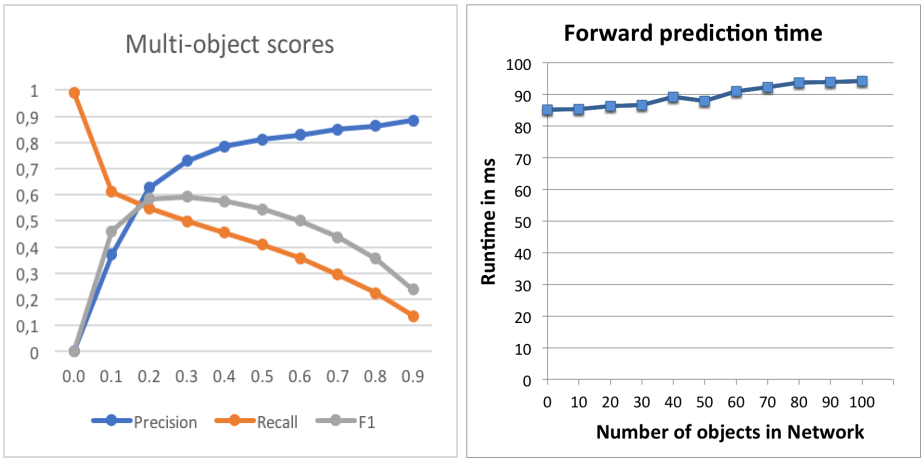


Figure 7.8: Left: Detection scores on the multi-object dataset for a different global threshold. Right: Runtime increase for the network prediction with an increased number of objects.

It is evident, however, that every additional parsed view V gives a larger benefit than taking more in-plane rotations R into the pool. We explain this by the fact that our viewpoint sampling is coarser than our in-plane sampling and thus reveals more uncovered pose space when parsed, which in turn helps especially depth-based refinement. Since we create a pool of $V \cdot R$ poses for each 2D detection, we fixed $V = 3, R = 3$ for all experiments as a compromise between accuracy and refinement runtime.

Performance on the two datasets We present our pose errors in Tables 7.3 and 7.4 after 2D and 3D refinement. Note that we do not compute the ADD scores for Tejani since each object is of (semi-)symmetric nature, leading always to near-perfect ADD scores of 1. The poses are visually accurate after 2D refinement and furthermore are boosted by an additional depth-based refinement stage. On the second dataset we are actually able to come very close to Brachmann *et al.* which is surprising since they have a huge advantage of real data training. For the case of pure RGB-based poses, we can even overtake their results. We provide more detailed error tables in the supplement.

7.4.2 Multiple object detection

The last dataset has annotations for 9 out of the 15 objects and is quite difficult since many instances undergo heavy occlusion. Different to the single object scenario, we have now a network with one global detection threshold for all objects and we present our scores in Figure 7.8 when varying this threshold. Brachmann *et al.* [Brachmann et al., 2016] can report an impressive Average Precision (AP) of 0.51 whereas we can report an AP of 0.38. It can be observed that our method degrades gracefully as the recall does not drop suddenly from one threshold step to the next. Note again that Brachmann *et al.* have the

Figure 7.9: One failure case where incorrect bounding box regression, induced by occlusion, led to wrong 6D hypothesis creation. In such cases a subsequent refinement cannot always recover the correct pose anymore.

advantage of training on real images of the sequence whereas we must detect heavily-occluded objects from synthetic training only.

7.4.3 Runtime and scalability

For a single object our method from the last chapter [Kehl et al., 2016b] has a runtime of around 650ms per frame whereas [Brachmann et al., 2016] report around 450ms. Above methods are scalable and thus have a sublinear runtime growth with an increasing database size. Our method is a lot faster than the related work while being scalable as well. In particular, we can report a runtime of approximately 85ms for a single object. We show our prediction times in Figure 7.8 which reveals that we scale very well with an increasing number of objects in the network. While the prediction is fast, our pose refinement takes more time since we need to refine every pose of each pool. On average, given that we have about 3 to 5 positive detections per frame, we need a total of an additional 24ms for refinement, leading to a total runtime of around 10Hz.

7.4.4 Failure cases

The most prominent issue is the difference in colors between synthetic model and scene appearance, also including local illumination changes such as specular reflections. In these cases, the object confidence might fall under the detection threshold since the difference between the synthetic and the real domain is too large. A more advanced augmentation would be needed to successfully tackle this problem. Another possible problem can stem from the bounding box regression. If the regressed corners are not providing a tight fit, it can lead to translations that are too offset during 6D pose construction. An example of this problem can be seen in Figure 7.9 where the occluded milk produces wrong offsets. We also observed that small objects are sometimes difficult to detect which is even more true after resizing the input to 299×299. Again, designing a more robust training as well as a larger network input could be of benefit here.

7.5 Conclusion

We have shown that color-based detectors are indeed able to match and surpass current state-of-the-art methods that leverage RGB-D data while being around one order of magnitude faster. Furthermore, the supplied 6D poses from color alone are sufficient for AR whereas further depth-based refinement provides perfect poses for manipulation purposes.

Chapter 8

Tracking via Joint Contour and Cloud Information

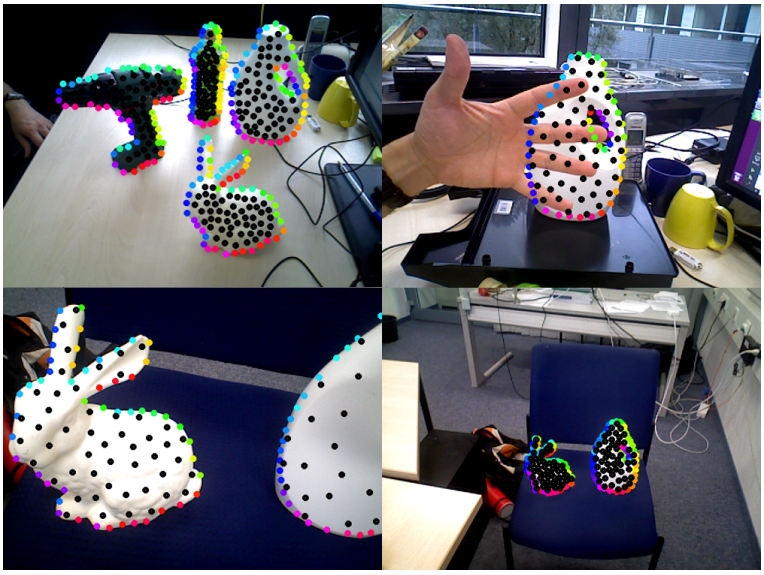


Figure 8.1: We can perform reliable tracking for multiple objects while being robust towards partial occlusions as well drastic changes in scale. To this end, we employ contour cues and interior object information to drive the 6D pose alignment jointly. All of the above is achieved on a single CPU core in real-time.

We propose a framework that allows accurate real-time tracking of multiple 3D models in color and depth. Unlike related works [Prisacariu and Reid, 2012, Tjaden et al., 2016] our method is lightweight, both in computation (requiring only one CPU core) and in memory footprint. To achieve this, we propose to prerender a given target 3D model from various viewpoints and extract occluding contour and interior information in an offline step. This avoids time consuming online renderings and consequently results in a fast tracking ap-

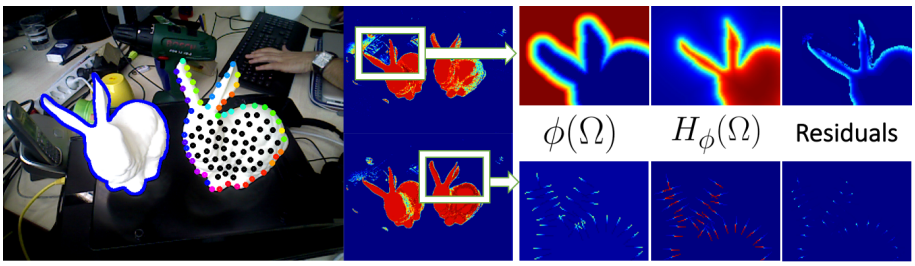


Figure 8.2: Tracking two Stanford bunnies side by side in color data. While the left is tracked densely, the right is tracked with a sparse set of 50 contour sample points. Starting from a computed posterior map P_f for each object, we evaluate all needed terms. The color on each sparse contour point represents its 2D gradient orientation. The black dots on the interior are used when having additional depth data.

proach. Furthermore, we do not compute the terms of our objective function densely but introduce sparse approximations which gives a tremendous performance boost, allowing real-time tracking of multiple instances. While the proposed contour-based tracking works well in RGB images, in the case of available depth information, we propose two additions: firstly, we make the color-based segmentation more robust by incorporating cloud information and secondly, we define a new tracking framework where a novel plane-to-point error on cloud data and a contour error are simultaneously steering the pose alignment.

- As a foundation of our work, we propose to prerender the model view space and extract contour and interior information in an offline step to avoid online rendering, making our method a pure CPU-based approach.
- We evaluate all terms sparsely instead of densely which gives a tremendous performance boost.
- Given RGB-D data, we show how to improve contour-based tracking by incorporating cloud information into the color contour estimation. Additionally, we present a new joint tracking that incorporates a novel plane-to-point error and a contour error, i.e. color and depth points are simultaneously steering the pose alignment.

Therefore, our method can deal with challenges typically encountered in tracking as depicted in the Figure 8.1. In the results section, we evaluate our approach both quantitatively and qualitatively and compare it to the related approaches reporting better accuracy at higher speeds.

8.1 Tracking via implicit contour embeddings

In the spirit of [Prisacariu and Reid, 2012, Tjaden et al., 2016] we want to track a (meshed) 3D model in camera space such that its projected silhouette aligns perfectly with a 2D segmentation in the image $I : \Omega \to \mathbb{R}^3$ with $\Omega \subset \mathbb{R}^2$ being

the image domain. Given a silhouette (i.e. foreground mask) $\Omega_f \subseteq \Omega$, we can infer a contour C to compute a 2D signed distance field (SDF) ϕ s.t.

$$\phi(\mathbf{x}) := \begin{cases} d(\mathbf{x}, C), & \text{if } \mathbf{x} \in \Omega_b \\ -d(\mathbf{x}, C), & \text{else} \end{cases}, \; d(\mathbf{x}, C) := \min_{\mathbf{y} \in C} ||\mathbf{x} - \mathbf{y}|| \tag{8.1}$$

where a pixel tells the signed distance to the closest contour point and $\Omega_b := \Omega \setminus \Omega_f$ is the set of background pixels.

We follow the PWP3D tracker energy formulation [Prisacariu and Reid, 2012] in which the pixel-wise (posterior) probability of a contour, embedded as ϕ, given color image I, is defined as

$$P(\phi|I) := \prod_{\mathbf{x} \in \Omega} \Big(H_\phi(\mathbf{x}) P_f(I(\mathbf{x})) + (1 - H_\phi(\mathbf{x}) P_b(I(\mathbf{x}))) \Big). \tag{8.2}$$

The terms P_f, P_b are modeling posterior distributions for foreground and background membership based on color, in practice computed from normalized RGB histograms, whereas H_ϕ represents a smoothed version of the Heaviside step function defined on ϕ. Assuming pixel-wise independence and taking the negative logarithm we get a functional

$$E := -\sum_{\mathbf{x} \in \Omega} \log \Big(H_\phi(\mathbf{x}) P_f(I(\mathbf{x})) + (1 - H_\phi(\mathbf{x}) P_b(I(\mathbf{x}))) \Big). \tag{8.3}$$

In order to optimize the energy in respect to a change in model pose, we employ a Gauss-Newton scheme over twist coordinates, similarly to Tjaden *et al.* [Tjaden et al., 2016]. As in the chapters before, we define a twist vector $\xi \in \mathbb{R}^6$ that provides a minimal representation for our sought transformation and its Lie algebra twist $\hat{\xi} \in se(3)$ as well as its exponential mapping to the Lie group $\Xi \in SE(3)$. We abuse notation s.t. $\Xi(\mathbf{X})$ expresses the transformation of $\Xi \in \mathbb{R}^{4\times4}$ applied to a 3D point $\mathbf{X}$. Assuming only infinitesimal change in transformation we derive the energy[1] in respect to a point $\mathbf{X}$ undergoing a screw motion ξ as

$$\frac{\partial E}{\partial \xi} = \frac{(P_f - P_b)}{H_\phi(P_f - P_b) + P_b} \frac{\partial H_\phi}{\partial \phi} \frac{\partial \phi}{\partial \mathbf{x}} \frac{\partial \pi(\mathbf{X})}{\partial \mathbf{X}} \frac{\partial \Xi(\mathbf{X})}{\partial \xi}. \tag{8.4}$$

A visualization of some terms can be seen in Figure 8.2. While $\frac{\partial \Xi(\mathbf{X})}{\partial \xi} \in \mathbb{R}^{3\times6}$ and $\frac{\partial \pi(\mathbf{X})}{\partial \mathbf{X}} \in \mathbb{R}^{2\times3}$ can be written in analytical form, $\frac{\partial H_\phi}{\partial \phi}$ resolves essentially to a smoothed Dirac delta whereas $\frac{\partial \phi}{\partial \mathbf{x}} \in \mathbb{R}^{1\times2}$ can be implemented via simple central differences. In total, we obtain one Jacobian $J_\mathbf{x} \in \mathbb{R}^{1\times6}$ per pixel and solve a least-squares problem

$$\nabla \xi = (\sum_{\mathbf{x}} J_\mathbf{x}^T J_\mathbf{x})^{-1} \sum_{\mathbf{x}} J_\mathbf{x} \tag{8.5}$$

via Cholesky decomposition. Given the model pose $M^t \in \mathbb{R}^{4\times4}$ at time t, we update via the exponential mapping

$$M^{t+1} = \exp(\hat{\nabla \xi}) \cdot M^t. \tag{8.6}$$

[1] For brevity, we moved the full derivation into appendix B.6

8.2 Approximating for winning

Computing the SDF from Eq. 8.1 has already three costly steps. We need a silhouette rendering Ω_f of the current model pose, an extraction of the contour C and lastly, a subsequent distance transform embedding ϕ. While [Tjaden et al., 2016] perform GPU rendering and couple computation of the SDF and its gradient in the same pass to be faster, [Prisacariu et al., 2015] perform hierarchical ray-tracing on the CPU and extract the contour via Scharr operators. We make two key observations:

1. Only the actual contour points are required
2. Neighboring points provide superfluous information because of similar curvature

We thus propose a cheap yet very effective approximation of the model render space that avoids both online rendering and contour extraction. In an offline stage, we equidistantly sample viewpoints V_i on a unit sphere around the object model, render from each and extract the 3D contour points to store view-dependent **sparse** 3D sampling sets in **local object space** (see Fig. 8.3). Since we will utilize these points in 3D space we neither need to sample in scale nor for different inplane rotations. Finally, we also store for each contour point its 2D gradient orientation and sample a set of interior surface points with their normals (see Fig. 8.4).

In a naive approach all involved terms from Eq. 8.3 would be computed densely, i.e. $\forall x \in \Omega$, which is prohibitively costly for real-time scenarios. The related work evaluates the energy only in a narrow band around the contour since the residuals decay quickly when leaving the interface. We therefore propose to compute Eq. 8.4 in a narrow band along a sparse set of selected contour points where we compute ϕ along rays. Each projected contour point shoots a positive and negative ray perpendicularly to the contour, i.e along its normal. Building on that, we introduce the idea of ray integration for 3D contour points such that we do not create pixel-wise but **ray-wise** Jacobians which leads to a smaller reduction step and a better conditioning of the normal system in Eq. 8.5 than [Prisacariu et al., 2015] and their approach.

To formalize, we have a model pose $[\mathbf{R}, \mathbf{T}]$ during tracking and avoid rendering by computing the camera position in object space $\mathbf{O} := -\mathbf{R}^T\mathbf{T}$. We normalize to unit length and find the closest viewpoint $\mathbf{V}^*$ quickly via dot products:

$$\mathbf{V}^* := \text{argmax}_{\mathbf{V}_i} \langle\ \mathbf{V}_i,\ \mathbf{O}/||\mathbf{O}||\ \rangle. \tag{8.7}$$

Each local 3D sample point of the contour $\mathbf{X}_i$ from V^* is then transformed and projected to a 2D contour sample point $\mathbf{x}_i = \pi(\mathbf{R}\mathbf{X}_i + \mathbf{T})$ which is then used for minimization of the error defined along the rays perpendicular to the object contour.

To get the orientation of each ray, we cannot rely anymore on the value during prerendering since the current model pose might have an inplane rotation not accounted for. Given a contour point with 2D rotation angle θ during prerendering, we could embed it into 3D space via $\mathbf{V} = (\cos\theta, \sin\theta, 0)$ and

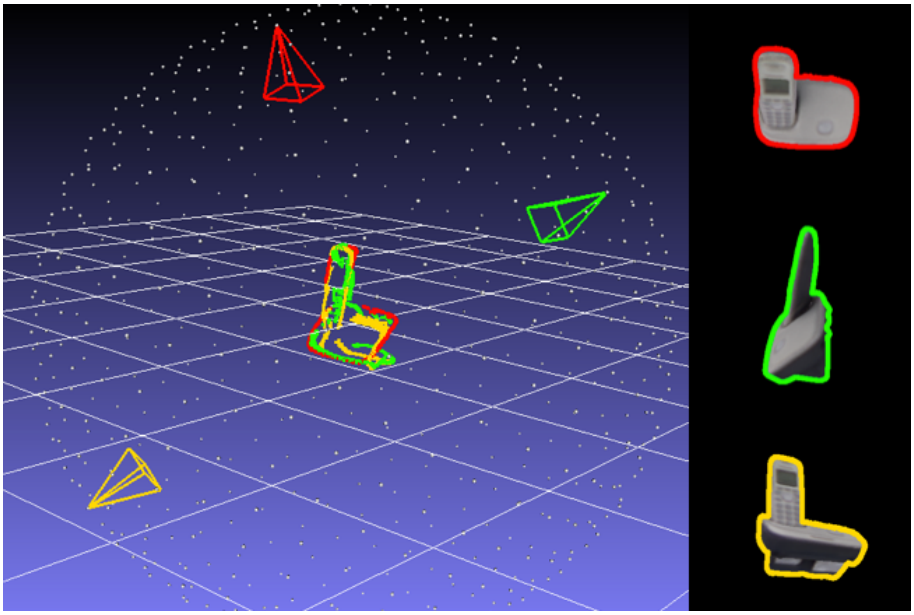


Figure 8.3: Object-local 3D contour points visualized for three viewpoints on the unit sphere. Each view captures a different contour which is used during tracking to circumvent costly renderings.

Figure 8.4: Current tracking and closest prerendered viewpoint augmented with contour and interior sampling points. The hue represents the normal orientation for each contour point. Note how we rotate the orientation of each contour point by our approximation of the inplane rotation such that the SDF computation is proper.

later multiply it with the current model rotation $\mathbf{R}$. Although this works in practice, the projection of $\mathbf{R} \cdot \mathbf{v}$ onto the image plane can be off at times. We thus propose a new approximation of the inplane rotation where we seek to decompose $\mathbf{R} = \mathbf{R}_{inplane} \cdot \mathbf{R}_{canonical}$ s.t. one part describes a general rotation around the object center in a canonical frame and the other a rotation around the view direction of the camera (i.e. inplane) . Although ill-posed in general, we exploit our knowledge about the closest viewpoint by assuming $\mathbf{R}_{canonical} \approx \mathbf{R}_{\mathbf{V}^*}$ and propose to approximate a rotation $\tilde{\mathbf{R}}$ on the xy-plane via

$$\tilde{\mathbf{R}} := \mathbf{R} \cdot \mathbf{R}_{\mathbf{V}^*}^T. \tag{8.8}$$

We then extract the angle $\theta = acos(\tilde{\mathbf{R}}_{1,1})$ via the first element. With larger viewpoint deviation $||\mathbf{V}^* - \frac{\mathbf{O}}{||\mathbf{O}||}||$ this approximation worsens but our sphere sampling is dense enough to alleviate this in practice. We re-orient each contour gradient $\tilde{\mathbf{g}_i} := (\mathbf{g}_i + \theta) \mod 2\pi$ and shoot rays to compute the residuals and $\frac{\partial H_\phi}{\partial \phi}$ from Eq. 8.3 (see Fig. 8.4 to compare the orientations and the bottom row in Figure 8.2 for the SDF rays).

The final missing building block is the derivative of the SDF $\frac{\partial \phi}{\partial \mathbf{x}}$ which cannot be computed numerically since we are missing dense information. We thus compute it geometrically, similar to [Prisacariu et al., 2015]. Whereas their computation is exact when assuming local planarity by projections onto the principal ray, our approach is faster while incurring a small error which is negligible in practice. Given a ray $\mathbf{r} = (r_x, r_y)$ from contour point $\mathbf{p} = (p_x, p_y)$ we compute the horizontal derivative at $\phi(p_x + r_x, p_y + r_y)$ as central difference

$$\frac{||(p_x + r_x + 1, p_y + r_y)|| - ||(p_x + r_x - 1, p_y + r_y)||}{2}. \tag{8.9}$$

The vertical derivative is computed analogously. Like the related work, we perform all computations on three pyramid levels in a coarse-to-fine manner and shoot the rays in a band of 8 steps on each level. Since we shoot two rays per contour point, our resulting normal system holds two ray Jacobians per point.

8.3 Extension to depth

We define a depth map $D : \Omega \to \mathbb{R}^+$ and its cloud map $\Pi_D^{-1} : \Omega \to \mathbb{R}^3$. Furthermore, we conduct a fast depth map inpainting such that we remove all unknown values in both D and Π_D^{-1}. Our goal is now two-fold: we want to make the posterior image P_f more robust by including cloud information into the probability estimation, and we want to extend the tracking energy to the new data.

8.3.1 Pixel-wise color/cloud posterior

Color histograms are very error prone in practice and fail quickly for textured/glossy objects and colorful backgrounds, even with adaptive histogram during tracking. We therefore propose a new robust pixel-wise posterior P_f to be used for contour alignment in Eq. 8.3 when additional depth data is provided.

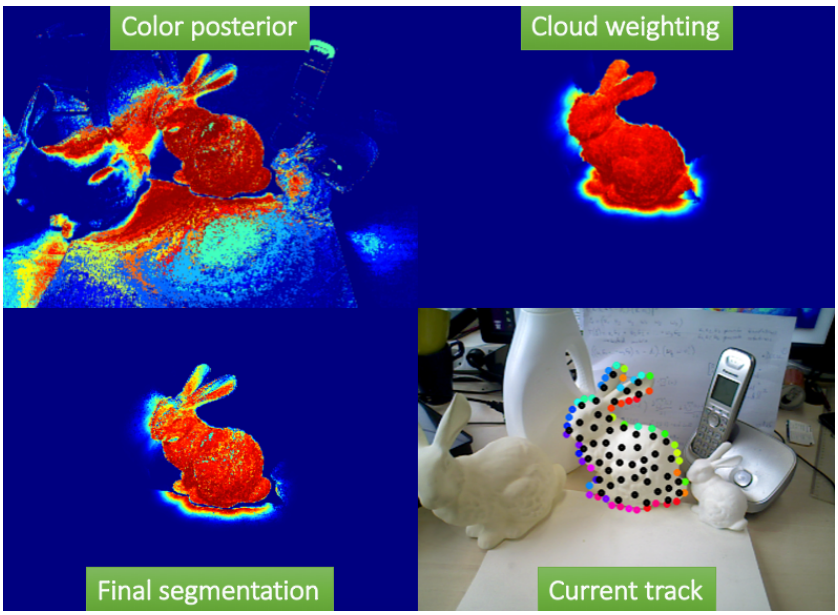


Figure 8.5: Segmentation computation. Since the background is similar in color, only the additional cloud-based weighting can give us a reliable segmentation to track against.

The notion we bring forward is that color posteriors alone are misleading and should be reweighed with their spatial proximity to the model. Given a model with pose $M = [\mathbf{R}, \mathbf{t}]$ we can infer silhouette region Ω_f and background Ω_b and want now to define their probabilities not only based on a given pixel color x but also an associated cloud point $\mathbf{C}$. We start from estimating the probability of a pose and its silhouette, provided color and cloud data, and make the harsh assumption that a pose and its silhouette are both independent:

$$P(\Omega_f, M|x, \mathbf{C}) := P(\Omega_f|x, \mathbf{C}) \cdot P(M|x, \mathbf{C}). \tag{8.10}$$

Assuming that all pixels are independent, that there is no correlation between the color of a pixel and its cloud point and that $P(M), P(\Omega_{\{f,b\}})$ are uniform, we reach:

$$P(\Omega_f|x, \mathbf{C}) := \frac{P(x|\Omega_f) \cdot P(\mathbf{C}|\Omega_f)}{\Sigma_{i\in\{f,b\}} P(x|\Omega_i) \cdot P(\mathbf{C}|\Omega_i)}, \tag{8.11}$$

$$P(M|x, \mathbf{C}) := \frac{P(x|M) \cdot P(\mathbf{C}|M)}{P(x, \mathbf{C}) = 1}. \tag{8.12}$$

We kindly refer to the appendix B.8 for the full derivation. While $P(x|\Omega_i)$ are usually computed from color histograms, it is not directly clear of how to compute $P(\mathbf{C}|\Omega_f)$ since it assumes inference for 3D data from an image mask while the term $P(x|M)$ is infeasible in general. We thus drop both terms (i.e. set to uniform) and finally define

$$P_f := P(\mathbf{C}|M) \cdot \frac{P(x|\Omega_f)}{\Sigma_{i\in\{f,b\}} P(x|\Omega_i)}. \tag{8.13}$$

The weighting term $P(\mathbf{C}|M)$, which gives the likelihood of a cloud point to be on the model, can be computed in multiple ways. Instead of simply taking the distance to the model centroid, we want a more precise measure that gives the distance to the **closest** model point. Since even logarithmic nearest-neighbor lookups would be costly here, we use an idea first presented in [Fitzgibbon, 2001]. One can precompute a distance transform in a volume around the model to facilitate a constant nearest-neighbor lookup function, $N(\mathbf{C}) := \arg\min_{\mathbf{X}\in Model} ||\mathbf{X}-\mathbf{C}||$, and we exploit this approach by bringing each scene cloud point $\mathbf{C}$ into the local object frame and efficiently calculate a pixel-wise weighting on the image plane with a Gaussian kernel:

$$P(\mathbf{C}|M) := \exp(-\frac{||\bar{\mathbf{C}} - N(\bar{\mathbf{C}})||}{\sigma^2}), \bar{\mathbf{C}} := \mathbf{R}^T \cdot \mathbf{C} - \mathbf{R}^T\mathbf{T}. \tag{8.14}$$

Here, $\sigma = 2.5cm$ steers how much deviation we allow a point to have from a perfect alignment since we want to deal with pose inacurracies as well as depth noise.

We can see the color posterior at work plus combination of the cloud-based weighting term in Figure 8.5. While the former gives a segmentation based on appearance alone, the latter takes complementary spatial distances into account, rendering contour-based tracking more robust.

8.3.2 Joint contour and cloud tracking

We want to introduce the notion of a combined tracking approach where 2D contour points and 3D cloud points are jointly driving the pose update. In essence, a weighted energy of the form

$$E_{Joint} = E_{Contour} + \lambda E_{ICP}. \tag{8.15}$$

where λ is supposed to balance both partial energies since they can deviate in the number of sample points as well as numerical scale.

In terms of ICP, a point-to-plane error has been shown to provide better and faster convergence then a point-to-point metric. It assumes alignment of source points s_i (here from a model view) to points d_i and normals n_i at the destination (here the scene). Normals in camera space can be approximated from depth images [Hinterstoisser et al., 2012a] but are usually noisy and require time. We thus propose a novel plane-to-point error where the normals are coming from the source point set and have been computed beforehand for each viewpoint. This ensures a fast runtime and perfect data alignment since tangent planes coincide at the optimum.

Given the current pose $[\mathbf{R}, \mathbf{T}]$ and closest viewpoint with local interior points $\mathbf{S}_i$, we transform to $\bar{\mathbf{S}}_i = \mathbf{R} \cdot \mathbf{S}_i + \mathbf{T}$ and project each to get the corresponding scene point $\mathbf{D}_i := \Pi_{\mathbf{D}}^{-1}(\pi(\bar{\mathbf{S}}_i))$. Since we also have a local $\mathbf{N}_i$ that we bring into

the scene, $\bar{\mathbf{N}}_i = \mathbf{R} \cdot \mathbf{N}_i$, we want to retrieve Ξ minimizing

$$E_{ICP} := \arg\min_{\Xi} \sum_i \Big((\Xi(\bar{\mathbf{S}}_i) - \mathbf{D}_i) \cdot \Xi_{SO}(\bar{\mathbf{N}}_i) \Big)^2. \tag{8.16}$$

The difference to the established point-to-plane error is solving for an additional rotation of the source normal $\bar{\mathbf{N}}_i$. Note that only the rotational part of Ξ acts on $\bar{\mathbf{N}}_i$ and we thus omit the translational generators of the Lie algebra. Deriving in respect to ξ[2], we get a Jacobian $J_i \in \mathbb{R}^{1\times 6}$ and a residual r_i for each correspondence

$$J_i := -\Big[\quad \bar{\mathbf{N}}_i^T \quad \Big((\bar{\mathbf{S}}_i \times \bar{\mathbf{N}}_i) + \bar{\mathbf{N}}_i \times (\bar{\mathbf{S}}_i - \mathbf{D}_i) \Big)^T \Big], \tag{8.17}$$

$$r_i := (\bar{\mathbf{S}}_i - \mathbf{D}_i) \cdot \bar{\mathbf{N}}_i \tag{8.18}$$

and construct a normal system to get a twist of the form

$$\nabla\xi = \Big(\sum_i J_i^T J_i \Big)^{-1} \sum_i J_i \cdot r_i \tag{8.19}$$

Altogether, we can now plug together Eqs. 8.3 and 8.19 to formulate the desired energy from Eq. 8.15 as a joint contour and plane-to-point alignment. Following up, we build a normal system that contains both the ray-wise contour Jacobians $J_{\mathbf{x}}$ from 2D image data and correspondence-wise ICP Jacobians J_i from 3D cloud data:

$$\nabla\xi = \Big(\sum_{\mathbf{x}} J_{\mathbf{x}}^T J_{\mathbf{x}} + \sum_i \lambda J_i^T J_i \Big)^{-1} \Big(\sum_{\mathbf{x}} J_{\mathbf{x}} + \sum_i \sqrt{\lambda} J_i \cdot r_i \Big). \tag{8.20}$$

Solving above system yields a twist with which the current pose is updated. The advantage of such a formulation is that we employ entities from different optimization problems into a common framework: while the color pixels minimize a projective error, the cloud points do so with a geometrical error. These complimentary cues can therefore compensate for each other if a segmentation is partially wrong or if some depth values are noisy.

8.4 Implementation details

We implemented our method in C++, running on a single CPU-core. In total, we render a model from equidistant 642 views, amounting to around 8 degrees in angular difference between two viewpoints. To compute the color histograms, we do not render to extract foreground/background masks. Instead, we simply fetch the colors at the projected interior points for the foreground histogram. For the background histogram, we compute the rectangular 2D projection of the model's 3D bounding box and take the pixels outside of it. We employ 1D lookup tables for both H_ϕ and its derivative to speed up computation. Lastly, if we find a projected transformed source point $\bar{\mathbf{S}}$ to be occluded, i.e. $D(\pi(\bar{\mathbf{S}}))+5cm < \bar{\mathbf{S}}_Z$, we neither set it as an ICP correspondence nor do we use it for the foreground histogram.

[2]The derivation can be found in the appendix B.7

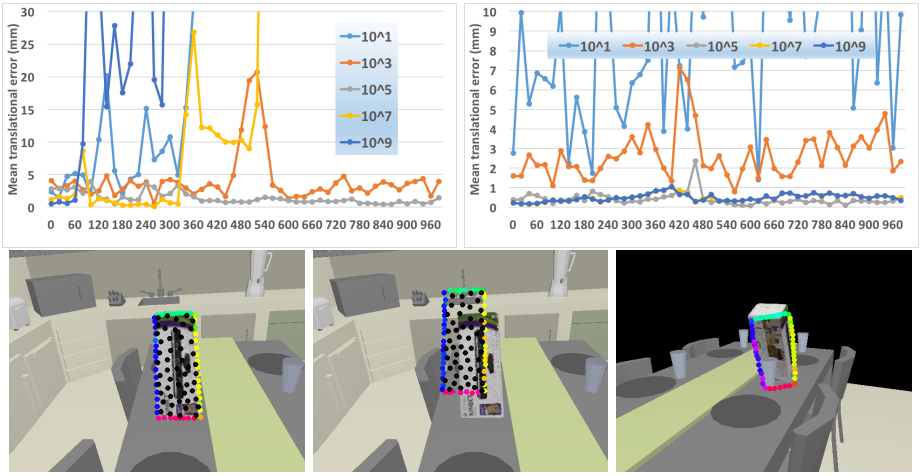


Figure 8.6: Top: Mean translational error for a changing λ on every 20th frame for 'kinect box' (left) and 'tide' (right). Bottom: Tracking performance on 'kinect box'. With $\lambda = 10^5$, the balance between contour and interior points drives the pose correctly. With $\lambda = 10^9$, the energy is dominated by the plane-to-point ICP term, which leads to drifting for planar objects. With an emphasis on contour alone ($\lambda = 10$), we deviate later due to an occluding cup.

8.5 Evaluation

To provide quantitative numbers and to self-evaluate our method on noise-free data, we run the first set of experiments on the synthetic RGB-D dataset of Choi and Christensen [Choi and Christensen, 2013]. It provides four sequences of 1000 frames where each covers an object around a given trajectory. Later, we run convergence experiments on the LineMOD dataset [Hinterstoisser et al., 2012a] and evaluate against Tan et al. on two of their sequences.

8.5.1 Balancing the tracking energy with λ

To understand the balancing between contour and interior points, we analyze the influence of a changing λ. It should both compensate for a different number of sampling points and numerical scale. We fix the sample points to 50 for both modalities to focus solely on the scale difference from the Jacobians. While the ICP values are metric, ranging around $[-1, 1]$, the values from the contour Jacobians are in image coordinates and can therefore be in the thousands. We chose two sequences, namely 'kinect box' and 'tide', and varied λ. All four sequences are impossible to track via contour alone ($\lambda = 10$) since the similarity between foreground and background is too large. On the other hand, relying on a plane-to-point energy alone ($\lambda = 10^9$) leads to planar drifting for the 'kinect box'. We therefore found $\lambda = 10^5$ to be a good compromise (see Figure 8.6).

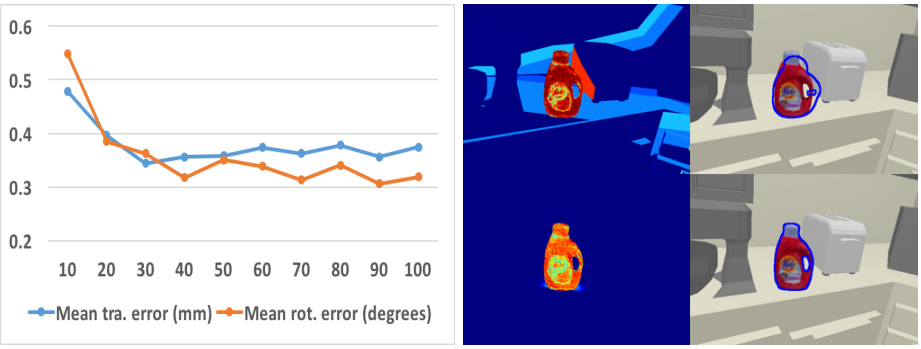


Figure 8.7: Left: Average error in translation/rotation for the 'tide' when varying the sample point size. We plot in the same chart since they are similar in scale. Right: Comparison of color posterior vs. cloud-reweighted when tracking with $\lambda = 10^3$.

8.5.2 Varying the number of sampling points

With a fixed $\lambda = 10^5$, we now look at the behavior when we change the number of sample points. We chose again the 'tide' since it has rich color and geometry to track against. As can be seen in Figure 8.7, we decrease constantly until 30 points where the translational error plateaus while the rotational error decays further, plateauing around 80-90 points. We were surprised to see that a rather small sampling set of 10 contour/interior points already leads to proper energy solutions, enabling successful tracking on the sequence.

8.5.3 Comparison to related work

We ran our method with $\lambda = 10^5$ and 50 points both on the contour and the interior. Since we wanted to measure the performance of the novel energy alignment with and without the additional cloud weighting, we repeated the experiments for both scenarios. We evaluate accordingly with the others by computing the RMSE on each translational axis as well as each rotational axis. As can be seen from Table 8.1, we outperform the other methods greatly, sometimes even up to one order of magnitude. This result is not really surprising, since we are the only method that does a direct, projective energy minimization. While both C&C and Krull use a particle filter approach that costs them more than 100ms, Tan evaluates a Random Forest based on depth differences. Tan and C&C employ depth information only whereas Krull uses RGB-D data like us.

If we compare our runtimes, we are very close to Tan. While they constantly need around 1.5ms, we need less than 3ms on average to compute the full update. If we compute the added cloud weighting, it takes us another 6ms but yields the lowest report error so far on this dataset. Note that both Tan and Krull require a training stage to build their regression structures whereas our method only needs to render 642 views and extract sample information. This takes about 5 seconds in total and requires roughly 10MB per model. Additionally, if we compare to the GPU-enabled dense implementation of Tjaden *et al.* [Tjaden et al., 2016],

we are roughly four times faster on a single CPU core.

8.5.4 Convergence properties

Since our proposed joint energy has not been applied in this manner before, we were curious about the general convergence behavior. To this end, we used the real-life LineMOD dataset [Hinterstoisser et al., 2012b]. Although designed for object detection, it has ground truth annotations for 15 textureless objects and we thus mimic a tracking scenario by perturbing the ground truth pose and 'tracking back' to the correct pose. More precisely, we create 1000 perturbations per frame by randomly sampling a perturbation angle in the range $[-\theta,\theta]$ separately for each axis and a random translational offset in the range $[-t,t]$ where t is $\frac{1}{10}th$ of the model's diameter. This yields **more than 1 million runs per sequence and configuration**, giving us a rigorous quantitative convergence analysis which we are presenting in Figure 8.8 on 3 sequences as histograms over the final rotational error. Additional results can be found in the appendix B.5.

We also plot the mean LineMOD score for each θ. For this, the model cloud is transformed once with the ground truth and with the retrieved pose and if the average Euclidean error between the two is smaller than $\frac{1}{10}th$ of the diameter, we count it as positive. Our optimization is iterative and coarse-to-fine on three levels and we thus computed above score for a different set of iterations. For example 2-2-1 indicates 2 iterations at the coarsest scale, 2 at the middle and 1 at the finest.

During tracking a typical change in pose rarely exceeds 5° on each axis and for this scenario, we can report near-perfect results. Nonetheless, we fare surprisingly well for more difficult pose deviations and degrade gracefully. From the LineMOD scores we see that one iteration on the finest level is not enough to recover stronger perturbations. For very high θ, the additional iterations on the coarser scales can make a difference in up to 10% which is mainly explained by the SDF rays, capturing larger spatial distances.

8.5.5 Real-data comparison to state of the art

We thank the authors from Tan *et al.* for providing two sequences together with ground truth annotation such that we could evaluate our algorithm in direct comparison to their method. In contrast to us, their method has a learned occlusion handling built-in. Both sequences feature a rotating table with a center object to track, undergoing many levels of occlusion. As can be seen from Figure 8.9 we outperform their approach, especially on the second sequence.

8.5.6 Failure cases

The weakest link in the method is the posterior computation since the whole contour energy is dependent on it. In the case of blur or a sudden change of colors (e.g. illumination) the posterior is misled. Furthermore, with our approximate SDF we sometimes fail for small or non-convex contours where the inner rays are overshooting the interior.

		PCL	C&C	Krull	Tan	A	B
(a) *Kinect Box*	t_x	43.99	1.84	0.8	1.54	1.2	**0.76**
	t_y	42.51	2.23	1.67	1.90	1.16	**1.09**
	t_z	55.89	1.36	0.79	0.34	**0.30**	0.38
	α	7.62	6.41	1.11	0.42	**0.14**	0.17
	β	1.87	0.76	0.55	0.22	0.23	**0.18**
	γ	8.31	6.32	1.04	0.68	0.22	**0.20**
	ms	4539	166	143	**1.5**	2.70	8.10
(b) *Milk*	t_x	13.38	0.93	**0.51**	1.23	0.91	0.64
	t_y	31.45	1.94	1.27	0.74	0.71	**0.59**
	t_z	26.09	1.09	0.62	**0.24**	0.26	**0.24**
	α	59.37	3.83	2.19	0.50	0.44	**0.41**
	β	19.58	1.41	1.44	**0.28**	0.31	0.29
	γ	75.03	3.26	1.90	0.46	0.43	**0.42**
	ms	2205	134	135	**1.5**	2.72	8.54
(c) *Orange Juice*	t_x	2.53	0.96	0.52	1.10	0.59	**0.50**
	t_y	2.20	1.44	0.74	0.94	**0.64**	0.69
	t_z	1.91	1.17	0.63	0.18	0.18	**0.17**
	α	85.81	1.32	1.28	0.35	**0.12**	**0.12**
	β	42.12	0.75	1.08	0.24	0.22	**0.20**
	γ	46.37	1.39	1.20	0.37	**0.18**	0.19
	ms	1637	117	129	**1.5**	2.79	8.79
(d) *Tide*	t_x	1.46	0.83	0.69	0.73	0.36	**0.34**
	t_y	2.25	1.37	0.81	0.56	0.51	**0.49**
	t_z	0.92	1.20	0.81	0.24	**0.18**	**0.18**
	α	5.15	1.78	2.10	0.31	0.20	**0.15**
	β	2.13	1.09	1.38	**0.25**	0.43	0.39
	γ	2.98	1.13	1.27	**0.34**	0.39	0.37
	ms	2762	111	116	**1.5**	2.71	9.42
Mean	Tra	18.72	1.36	0.82	0.81	0.58	**0.51**
	Rot	29.70	2.45	1.38	0.37	0.28	**0.26**
	ms	2786	132	131	**1.5**	2.73	8.71

Table 8.1: Errors in translation (mm) and rotation (degrees), and the runtime (ms) of the tracking results on the Choi dataset. We compare PCL's ICP, C&C [Choi and Christensen, 2013], Krull *et al.* [Krull et al., 2014] and Tan *et al.* [Tan et al., 2015] to us without (A) and with cloud weighting (B).

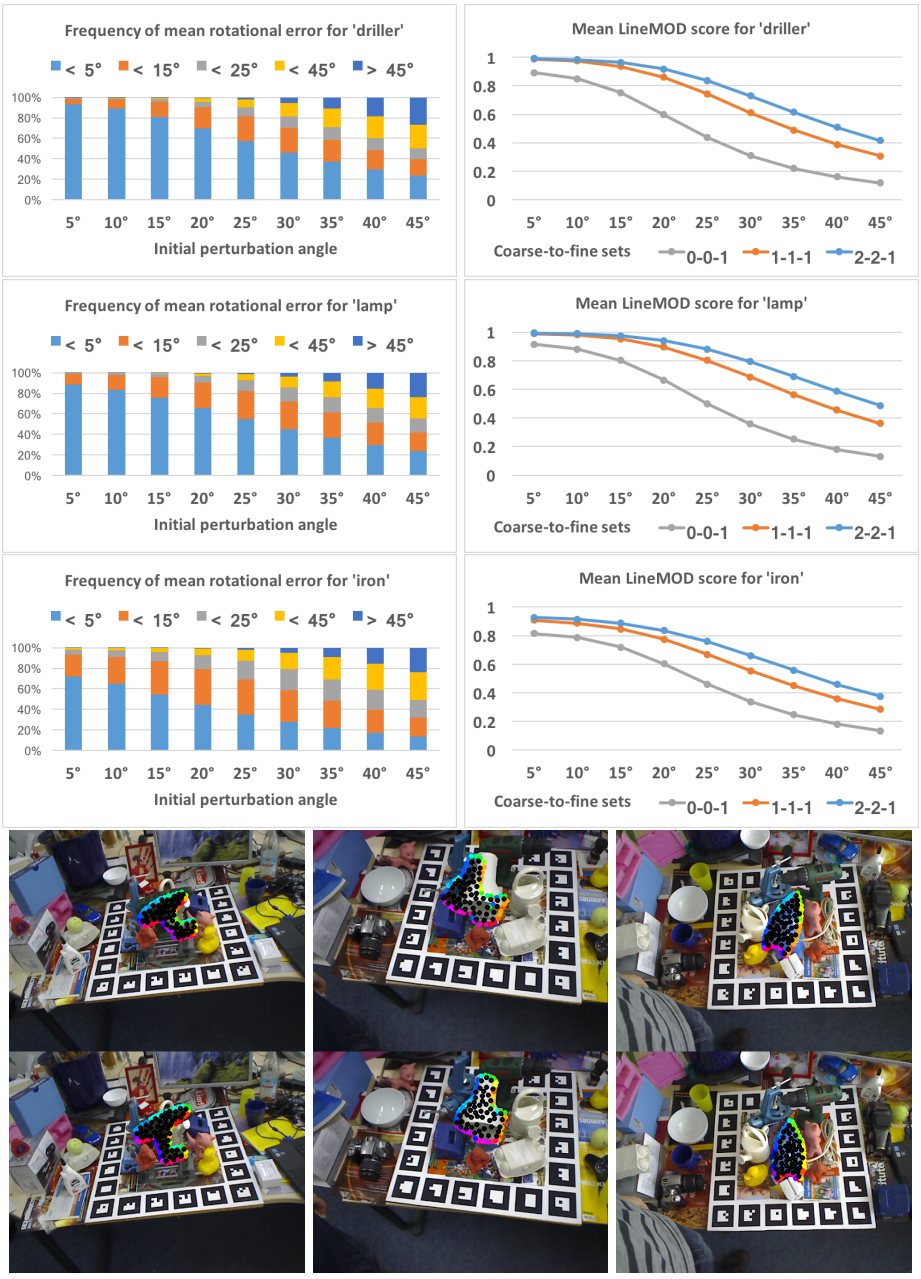


Figure 8.8: Top: Relative frequency of rotational error for each θ. Center: Mean LineMOD scores for each θ and a given iteration scheme. Bottom: Perturbation examples and retrieved poses.

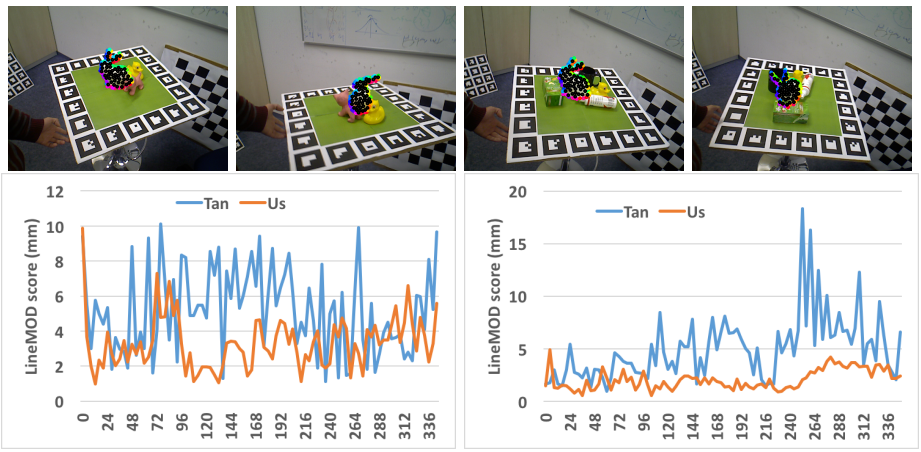


Figure 8.9: Top: Two frames each from the two sequences that we compared against Tan *et al.* . Bottom: The LineMOD error for every $4th$ frame on both sequences. We clearly perform better.

8.6 Conclusion

We have demonstrated how RGB and depth can be utilized in a joint fashion for the goal of accurate and efficient 6D pose tracking. The proposed algorithm relies on a novel optimization scheme that is general enough to be individually applied on either the depth or the RGB modality, while being able to fuse them in a principled way when both are available. Our system runs in real-time using a single CPU core, and can track around 10 objects at 30Hz, which is a realistic upper bound on what can visually fit into one VGA image. At the same time, it is able to report state-of-the-art accuracy and inherent robustness towards occlusion.

Chapter 9

Conclusion

Here we will present a summary of the thesis contributions, stress their importance as well as their limitations and will discuss possible paths for future research.

9.1 Summary

For the domains of 3D object reconstruction as well as 3D detection, 6D pose estimation and tracking, we have presented important contributions to the state of the art. For 3D object reconstruction, we have shown how to recover richly textured, metrically accurate 3D object models in a table top scenario. As a key contribution, we have identified that optimization must be incorporated in each single step of the pipeline to achieve the best results. Optimizing over the camera poses to achieve proper alignment between views, as well as improving the actual range image integration, have resulted in a significant increase in overall fidelity compared to baseline methods. Since the integration of more views can easily exhaust memory constraints, we have provided an alternative optimization scheme for range data fusion that employs a more efficient space partitioning.

The second part of the thesis has introduced multiple novel ways to further the domain of simultaneous object detection and 6D pose estimation. Starting from a traditional template matching approach, we have provided an improved variant via learned hashing functions that outperforms the closest competitor and allows for scalability. We then switched over to methods that are based on Deep Learning and contributed two orthogonal approaches: 1) a local voting-based variant that uses RGB-D imagery in conjunction with codebooks, leveraging CNNs for feature extraction and scene matching, and 2) a holistic RGB-only variant that brings the single-shot detection framework over the domain of 6D estimation. Both variants have greatly improved over the compared baselines and provide two distinct views on how object detection can be accomplished. Last but not least, we have contributed a lightweight, scalable 6D pose tracking formalism that could be either used as a stand-alone method or in conjunction with our proposed object detectors. The tracker introduces a novel, complimen-

tary energy over contours and interior surface area and has proven robustness to many typical challenges such as occlusion and drastic changes in scale and illumination.

9.2 Limitations and Future Work

One distinct limitation of our RGB-D based reconstruction pipeline is the requirement of static geometry. The setup in its current form could not cope well with strong dynamic changes in the scene and requires a table top for proper background/foreground separation. Additionally, our current formulation does not allow reconstruction of non-rigid (either articulate or deformable) objects. This aspect has been addressed by others, for example [Fujiwara et al., 2011] by assuming local rigidity, tracking shape templates [Zollhöfer et al., 2014] or deformation fields [Innmann et al., 2016]. While these topics come with their own difficulties, they allow capturing semantically-richer object categories. Nonetheless, our assumption is valid for most man-made objects in the environment and we have provided multiple examples throughout the thesis.

Connected with the static assumption is also the second part of our thesis, since the presented detection as well as tracking methods rely on static model geometry. Our given formulations require that an object pose provides a 1:1 mapping to a visual appearance or view, and breaking this mapping would result in ambiguity not handled by our approaches. There are research avenues that explore deformable 3D detection, e.g. [Fidler et al., 2012, Drost and Ilic, 2015], but the additional degrees of freedom usually have a negative impact on pose accuracies since those two quantities can be ambiguously coupled. In the work of [Manhardt et al., 2019a] the authors learn to express ambiguity by predicting pose distributions. Another research path which will probably see much more focus is synthetic training for learning deep models. We used synthetic data for training our detectors since a synthetic data pipeline can, in theory, provide for an infinite source of annotated images. Especially for pure RGB-based approaches, effective model training from solely synthetic data still eludes understanding and our single-shot detector needed ImageNet [Russakovsky et al., 2015] pretraining as well real background images for proper generalization. Recent research efforts have shown that reducing the domain gap is indeed possible via different techniques ([Tobin et al., 2017, Tremblay et al., 2018, Zakharov et al., 2019]) and we can expect more efficient methods to be discovered soon.

Appendix A

Authored and Co-authored Publications

Authored:

1. Kehl, W., Navab, N., and Ilic, S. (2014). Coloured signed distance fields for full 3D object reconstruction. In *BMVC*

2. Kehl, W., Tombari, F., Navab, N., Ilic, S., and Lepetit, V. (2015). Hashmod: A Hashing Method for Scalable 3D Object Detection. In *BMVC*

3. Kehl, W., Milletari, F., Tombari, F., Ilic, S., and Navab, N. (2016b). Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation. In *ECCV*

4. Kehl, W., Holl, T., Tombari, F., Ilic, S., and Navab, N. (2016a). An Octree-Based Approach towards Efficient Variational Range Data Fusion. In *BMVC*

5. Kehl, W., Tombari, F., Ilic, S., and Navab, N. (2017b). Real-Time 3D Model Tracking in Color and Depth on a Single CPU Core. In *CVPR*

6. Kehl, W., Manhardt, F., Ilic, S., Tombari, F., and Navab, N. (2017a). SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. In *ICCV*

7. Manhardt, F., Kehl, W., Navab, N., and Tombari, F. (2018). Deep Model-Based 6D Pose Refinement in RGB. In *ECCV*

8. Manhardt, F., Kehl, W., and Gaidon, A. (2019b). ROI-10D: Monocular Lifting of 2D Detection to 6D Pose and Metric Shape. In *CVPR*

Co-authored:

1. Habert, S., Meng, M., Kehl, W., Wang, X., Tombari, F., Fallavollita, P., and Navab, N. (2015). Augmenting mobile C-arm fluoroscopes via Stereo-RGBD sensors for multimodal visualization. In *ISMAR*

2. Milletari, F., Kehl, W., Tombari, F., , Ilic, S., Ahmadi, A., and Navab, N. (2015). Universal Hough dictionaries for object tracking. In *BMVC*

3. Slavcheva, M., Kehl, W., Navab, N., and Ilic, S. (2016). SDF-2-SDF: Highly Accurate 3D Object Reconstruction. *ECCV*

4. Slavcheva, M., Kehl, W., Navab, N., and Ilic, S. (2017). SDF-2-SDF Registration for Real-Time 3D Reconstruction from RGB-D Data. *IJCV*

5. Zakharov, S., Kehl, W., Planche, B., Hutter, A., and Ilic, S. (2017). 3D object instance recognition and pose estimation using triplet loss with dynamic margin. In *IROS*

6. Nissler, C., Badshah, I., Castellini, C., Kehl, W., and Navab, N. (2017). Improving Optical Myography via Convolutional Neural Networks. In *MEC*

7. Hodan, T., Michel, F., Brachmann, E., Kehl, W., Buch, A., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., Sahin, C., Manhardt, F., Tombari, F., Kim, T.-K., Matas, J., and Rother, C. (2018). BOP: Benchmark for 6D Object Pose Estimation. In *ECCV*

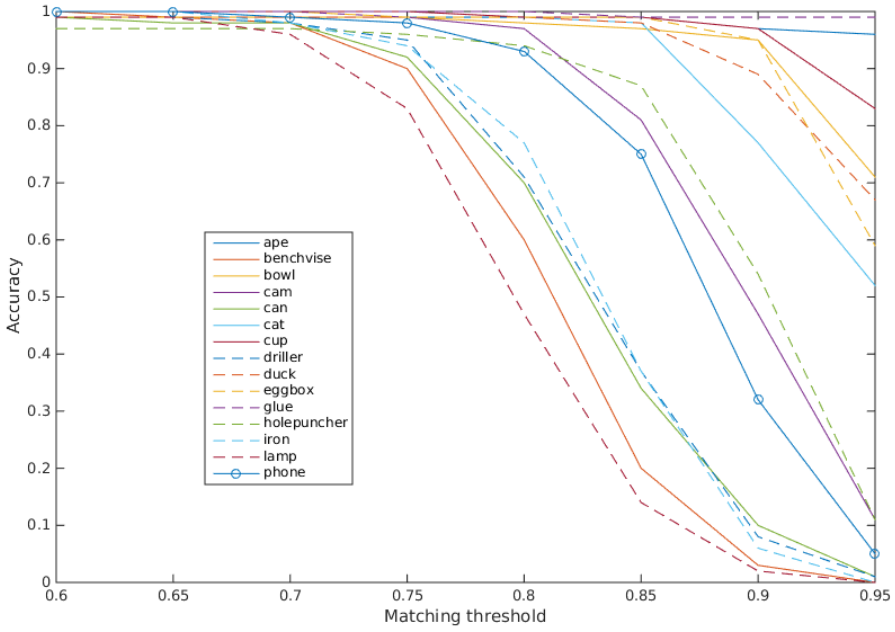
8. Zakharov, S., Kehl, W., and Ilic, S. (2019). DeceptionNet: Network-Driven Domain Randomization. In *ICCV*

Appendix B

Additional results

B.1 Matching thresholds for Hashmod

In order to determine each object-dependent matching threshold for our grid-based similarity score, we conducted a sweep over the thresholds for each sequence and fixed them such that each object would be found at least with an average accuracy of 98% in an exhaustive search scenario.



B.2 More results for the Single-Shot Detector

Object-wise detection scores We present the detection score graphs for each object of the first two datasets in Figures B.1 and B.2 from which we determined the best object-wise threshold, listed in Table B.1.

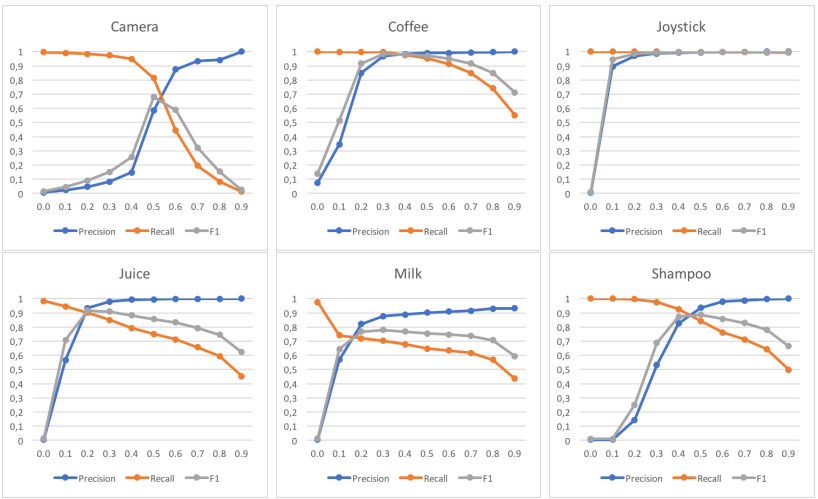


Figure B.1: Threshold sweeping the detection scores for each 'Tejani' object.

Camera	Coffee	Joystick	Juice	Milk	Shampoo
0.55	0.35	0.5	0.25	0.3	0.45

ape	bvise	cam	can	cat	driller	duck	box
0.5	0.15	0.2	0.75	0.35	0.25	0.25	0.25

glue	holep	iron	lamp	phone
0.4	0.4	0.3	0.55	0.35

Table B.1: Object-wise thresholds for the Tejani dataset (upper row) and the LineMOD dataset.

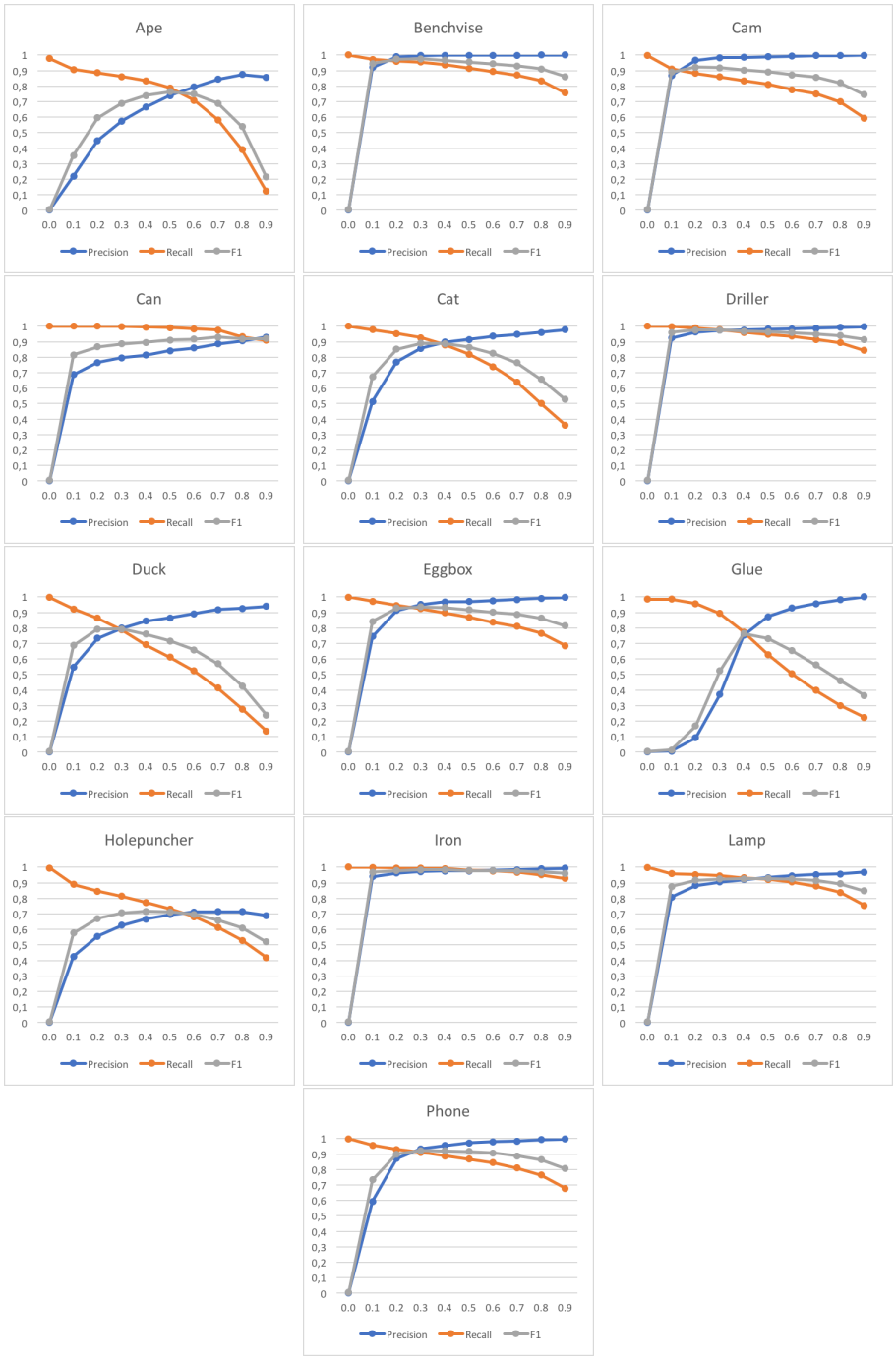


Figure B.2: Threshold sweeping the detection scores for each 'LineMOD' object.

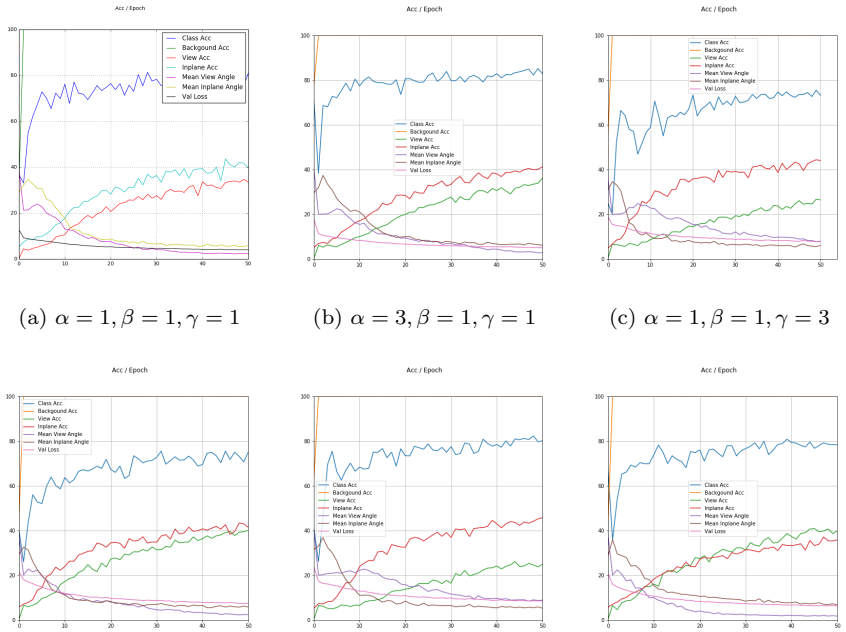
B.3 Detailed pose errors for the LineMOD dataset

	ape	bvise	cam	can	cat	driller	duck	box	glue	holep	iron	lamp	phone
IoU-2D	0.99	1.00	0.99	1.0	0.99	0.99	0.98	0.99	0.98	0.99	0.99	0.99	1.00
IoU-3D	0.96	0.98	0.98	0.99	0.95	0.95	0.95	0.98	0.89	0.97	0.97	0.98	0.93
VSS-2D	0.73	0.67	0.73	0.75	0.67	0.66	0.71	0.78	0.72	0.70	0.74	0.66	0.72
VSS-3D	0.84	0.88	0.90	0.86	0.81	0.84	0.83	0.88	0.75	0.77	0.85	0.84	0.81
ADD-2D	0.65	0.80	0.78	0.86	0.70	0.73	0.66	1.00	1.00	0.49	0.78	0.73	0.79
ADD-3D	0.85	0.94	0.94	0.94	0.86	0.85	0.82	1.00	1.00	0.73	0.95	0.87	0.87

Table B.2: Object-wise pose errors for the LineMOD dataset.

B.4 Errors for different loss term weighting

We plot the average error on a synthetic validation set. While the accuracies for class, viewpoint and in-plane rotations increase, the networks converge at different levels. We also plot the more important mean angular deviation for viewpoint and in-plane rotation since this is usually the expected error of the pooled hypotheses before refinement.



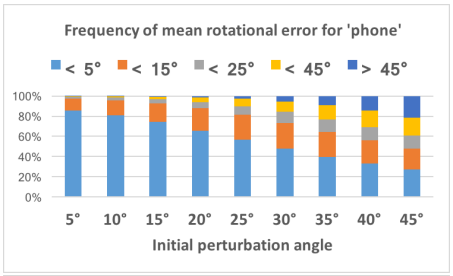
(a) $\alpha = 1, \beta = 1, \gamma = 1$ (b) $\alpha = 3, \beta = 1, \gamma = 1$ (c) $\alpha = 1, \beta = 1, \gamma = 3$

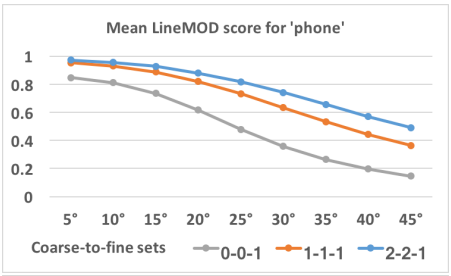
(d) $\alpha = 1, \beta = 2, \gamma = 2$ (e) $\alpha = 3, \beta = 1, \gamma = 3$ (f) $\alpha = 2, \beta = 2, \gamma = 1$

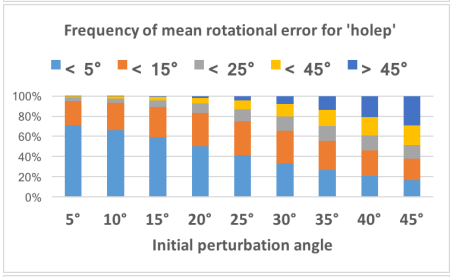
Figure B.3: Development of training errors on a synthetic validation set.

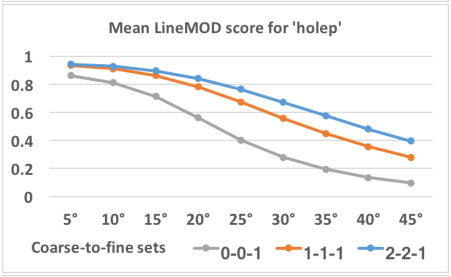
B.5 Tracker convergence results

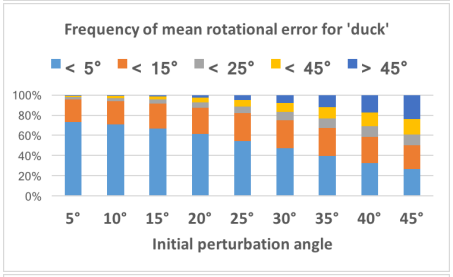
We run the experiments on all non-symmetric objects of the LineMOD dataset since measure for symmetric objects is misleading and accepts wrong poses. As in the main paper, the degradation for larger deviations is increasing but we do not see a sharp sudden decline. Again, additional iterations on multiple levels drastically improves the general alignment under all perturbations.

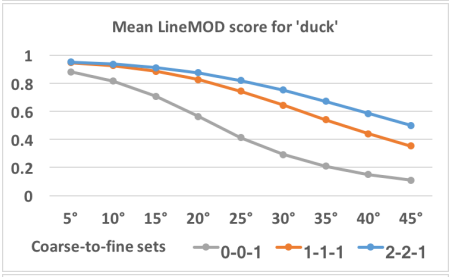


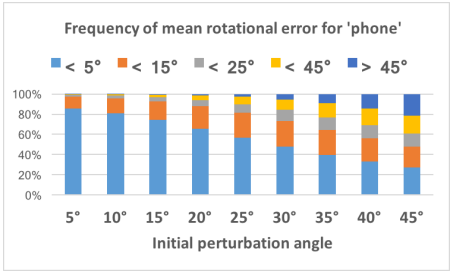


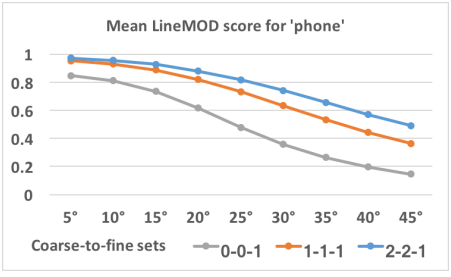


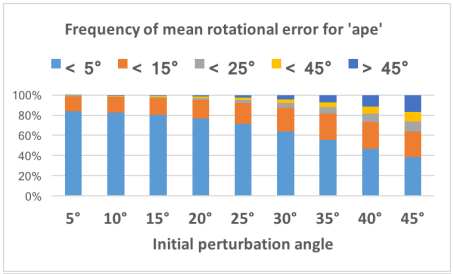


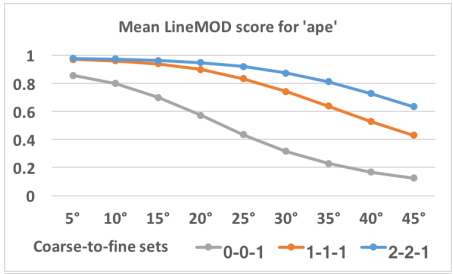


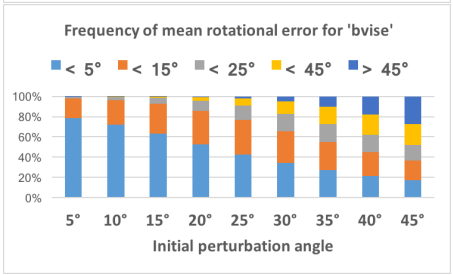


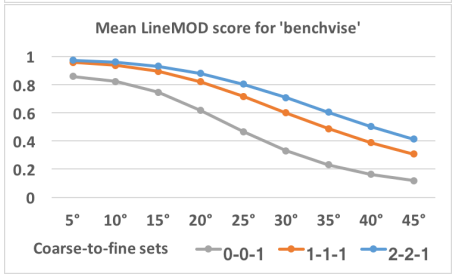


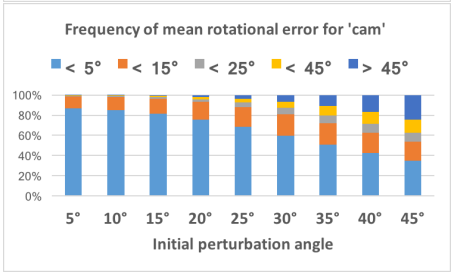


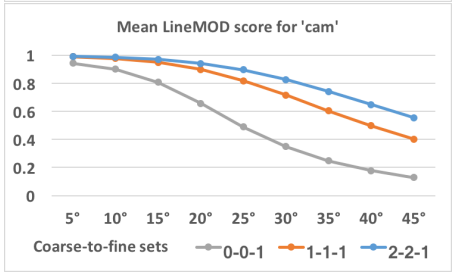

Frequency of mean rotational error for 'ape'
< 5° < 15° < 25° < 45° > 45°
100%
80%
60%
40%
20%
0%
5° 10° 15° 20° 25° 30° 35° 40° 45°
Initial perturbation angle

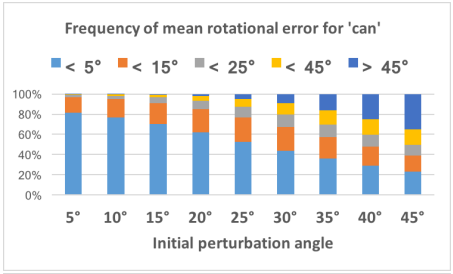

Mean LineMOD score for 'ape'
1
0.8
0.6
0.4
0.2
0
5° 10° 15° 20° 25° 30° 35° 40° 45°
Coarse-to-fine sets
0-0-1
1-1-1
2-2-1

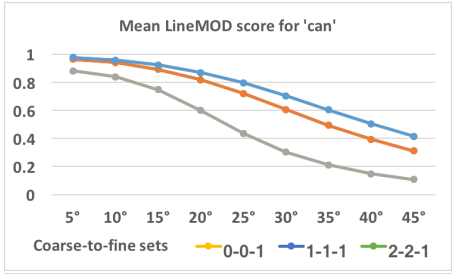

Frequency of mean rotational error for 'bvise'
< 5° < 15° < 25° < 45° > 45°
100%
80%
60%
40%
20%
0%
5° 10° 15° 20° 25° 30° 35° 40° 45°
Initial perturbation angle

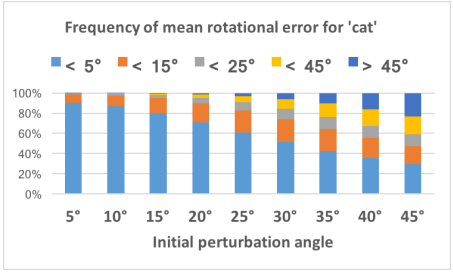

Mean LineMOD score for 'benchvise'
1
0.8
0.6
0.4
0.2
0
5° 10° 15° 20° 25° 30° 35° 40° 45°
Coarse-to-fine sets
0-0-1
1-1-1
2-2-1

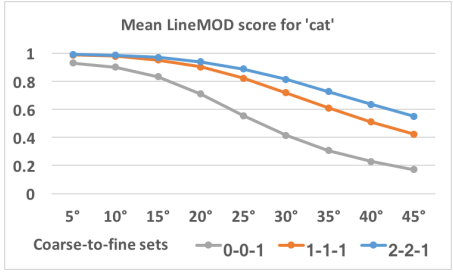

Frequency of mean rotational error for 'cam'
< 5° < 15° < 25° < 45° > 45°
100%
80%
60%
40%
20%
0%
5° 10° 15° 20° 25° 30° 35° 40° 45°
Initial perturbation angle


Mean LineMOD score for 'cam'
1
0.8
0.6
0.4
0.2
0
5° 10° 15° 20° 25° 30° 35° 40° 45°
Coarse-to-fine sets
0-0-1
1-1-1
2-2-1


Frequency of mean rotational error for 'can'
< 5° < 15° < 25° < 45° > 45°
100%
80%
60%
40%
20%
0%
5° 10° 15° 20° 25° 30° 35° 40° 45°
Initial perturbation angle


Mean LineMOD score for 'can'
1
0.8
0.6
0.4
0.2
0
5° 10° 15° 20° 25° 30° 35° 40° 45°
Coarse-to-fine sets
0-0-1
1-1-1
2-2-1


Frequency of mean rotational error for 'cat'
< 5° < 15° < 25° < 45° > 45°
100%
80%
60%
40%
20%
0%
5° 10° 15° 20° 25° 30° 35° 40° 45°
Initial perturbation angle


Mean LineMOD score for 'cat'
1
0.8
0.6
0.4
0.2
0
5° 10° 15° 20° 25° 30° 35° 40° 45°
Coarse-to-fine sets
0-0-1
1-1-1
2-2-1

B.6 Optimization of the Contour Energy

Starting from the probability of a contour ϕ when given an observed image I

$$P(\phi|I) := \prod_{\mathbf{x}\in\Omega} \Big(H_\phi(\mathbf{x}) P_f(I(\mathbf{x})) + (1 - H_\phi(\mathbf{x}) P_b(I(\mathbf{x}))) \Big) \tag{B.1}$$

that we seek to maximize, one can instead minimize the negative log which breaks down to a sum of pixel-wise terms:

$$E_C := -\sum_{\mathbf{x}\in\Omega} \log \Big(H_\phi(\mathbf{x}) P_f(I(\mathbf{x})) + (1 - H_\phi(\mathbf{x}) P_b(I(\mathbf{x}))) \Big). \tag{B.2}$$

The derivation in respect to a change in pose

$$\frac{\partial E_C}{\partial \xi} = -\frac{(P_f - P_b)}{H_\phi(P_f - P_b) + P_b} \frac{\partial H_\phi}{\partial \phi} \frac{\partial \phi}{\partial \mathbf{x}} \frac{\partial \pi(\mathbf{X})}{\partial \mathbf{X}} \frac{\partial \Xi(\mathbf{X})}{\partial \xi}. \tag{B.3}$$

can be then expressed via the following terms. Firstly, the smoothed Heaviside H_ϕ and its derivative $\frac{\partial H_\phi}{\partial \phi}$ as a smoothed Dirac delta

$$H_\phi(x) := \frac{1}{\pi}\Big(-atan(b \cdot x) + \frac{\pi}{2} \Big) \qquad \frac{\partial H_\phi}{\partial \phi}(x) := \frac{1}{\pi(x^2 + b^2 + 1)} \tag{B.4}$$

with $b = 0.5$ being the grade of applied smoothing to the contour embedding in our implementation. Provided the 3D point $\mathbf{X} = (X_x, X_y, X_z)$ to the projected 2D point $\mathbf{x}$ and intrinsics f_x, f_y, we write the rest of the derivatives as

$$\frac{\partial \pi(\mathbf{X})}{\partial \mathbf{X}} = \begin{bmatrix} \frac{f_x}{X_z} & 0 & \frac{f_x \cdot X_x}{X_z^2} \\ 0 & \frac{f_y}{X_z} & \frac{f_y \cdot X_y}{X_z^2} \end{bmatrix} \qquad \frac{\partial \Xi(\mathbf{X})}{\partial \xi} = \begin{bmatrix} 1 & 0 & 0 & 0 & X_z & -X_y \\ 0 & 1 & 0 & -X_z & 0 & X_x \\ 0 & 0 & 1 & X_y & -X_x & 0 \end{bmatrix}. \tag{B.5}$$

Given with the other explanations from the paper, we can now compute the Jacobian $J_\mathbf{x} := \frac{\partial E_C}{\partial \xi}(x\mathbf{x})$ and update the pose as explained via Cholesky decomposition and the exponential map.

B.7 Optimization of the Plane-to-Point Energy

Given source points $\mathbf{S}_i$ and source normals $\mathbf{N}_i$, we seek an alignment to destination points d_i such that

$$E_{ICP} := \arg\min_{\Xi} \sum_i \Big((\Xi(\mathbf{S}_i) - \mathbf{D}_i) \cdot \Xi_{SO}(\mathbf{N}_i) \Big)^2 . \tag{B.6}$$

Deriving in respect to ξ for a given correspondence i yields us

$$\frac{\partial E_{ICP}}{\partial \xi}(\mathbf{S}_i, \mathbf{N}_i, \mathbf{D}_i, \xi) = 2 \cdot \left((\Xi(\mathbf{S}_i) - \mathbf{D}_i) \cdot \frac{\partial \Xi_{SO}}{\partial \xi}(\mathbf{N}_i) + \frac{\partial (\Xi(\mathbf{S}_i) - \mathbf{D}_i)}{\partial \xi} \cdot \Xi_{SO}(\mathbf{N}_i) \right) = \tag{B.7}$$

$$2 \cdot \left((\mathbf{S}_i - \mathbf{D}_i) \cdot \begin{bmatrix} 0 & 0 & 0 & 0 & N_z & -N_y \\ 0 & 0 & 0 & -N_z & 0 & N_x \\ 0 & 0 & 0 & N_y & -N_x & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 & S_z & -S_y \\ 0 & 1 & 0 & -S_z & 0 & S_x \\ 0 & 0 & 1 & S_y & -S_x & 0 \end{bmatrix} \cdot \mathbf{N}_i \right) \tag{B.8}$$

Starting from here, we want to employ a Gauss-Newton scheme for the optimization. We thus seek an increment $\Delta\xi$ around $\mathbf{0}$ such that we minimize the error, i.e. we conduct Taylor expansion around zero s.t.

$$E_{ICP}(\mathbf{0} + \Delta\xi) = E_{ICP}(\mathbf{0}) + \frac{\partial E_{ICP}}{\partial \xi}(\mathbf{0}) \cdot \Delta\xi + \frac{1}{2!} \left(\frac{\partial E^2_{ICP}}{\partial \xi}(\mathbf{0}) \cdot \Delta\xi \right) \cdot \Delta\xi + h.o.t. \tag{B.9}$$

Following the typical approximation scheme, we disregard the higher order terms. Defining the residual $r_i := (\mathbf{S}_i - \mathbf{D}_i) \cdot \mathbf{N}_i$ and a Jacobian $J_i := [\, \mathbf{N}_i^T \quad \big((\mathbf{S}_i - \mathbf{D}_i) \times \mathbf{N}_i + \mathbf{S}_i \times \mathbf{N}_i \big)^T]$, we arrive at

$$E_{ICP}(\mathbf{0} + \Delta\xi) \approx r_i^2 + \Delta\xi^T J_i^T r_i + \Delta\xi^T J_i^T J_i \Delta\xi. \tag{B.10}$$

To minimize $E_{ICP}(\mathbf{0} + \Delta\xi)$, we can now derive in respect to $\Delta\xi$ and set it to zero to find the best update $\Delta\xi$:

$$\frac{\partial E_{ICP}(\mathbf{0} + \Delta\xi)}{\partial \Delta\xi} = J_i^T r_i + J_i^T J_i \Delta\xi \stackrel{!}{=} 0 \Leftrightarrow \Delta\xi = -(J_i^T J_i)^{-1} J_i^T r_i. \tag{B.11}$$

Finally, to fuse it seamlessly into the contour optimization, we negate J_i (or alternatively r_i) and retrieve the final normal system for the joint energy.

B.8 Derivation of the Posterior

What we essentially want to formulate is the probability of a model pose $M = [R, t]$ and its projected silhouette Ω_f, given color image I and cloud data Π^{-1},

$$P(\Omega_f, M|I, \Pi^{-1}). \tag{B.12}$$

This expression is difficult to compute in general since it involves 2D and 3D entities as well as a 6D pose. Instead, we make the first step towards tractability by assuming that each pixel is independent. We thus rephrase Ω_f as a binary variable $G \in \{FG, BG\}$ that signifies whether a certain pixel is foreground or background. Dealing now with pixel-wise colors x and cloud points $\mathbf{C}$, we also assume that a pose and its projected silhouette are independent entities, given x and C:

$$P(G = FG, M|x, \mathbf{C}) = P(FG|x, \mathbf{C}) \cdot P(M|x, \mathbf{C}). \tag{B.13}$$

We now go into the derivation of those two terms. Applying Bayes' rule, we get

$$P(FG|x, \mathbf{C}) := \frac{P(x, \mathbf{C}|FG) \cdot P(FG)}{P(x, \mathbf{C})} \qquad P(M|x, \mathbf{C}) := \frac{P(x, \mathbf{C}|M) \cdot P(M)}{P(x, \mathbf{C})} \tag{B.14}$$

and we now assume further that colors and cloud points are independent, given the foreground or pose model. From here, we marginalize over both instances of G as well as the model pose space:

$$P(FG|x, \mathbf{C}) := \frac{P(x|FG) \cdot P(\mathbf{C}|FG) \cdot P(FG)}{\Sigma_{G \in \{FG, BG\}} P(x|G) \cdot P(\mathbf{C}|G) \cdot P(G)} \tag{B.15}$$

$$P(M|x, \mathbf{C}) := \frac{P(x|M) \cdot P(\mathbf{C}|M) \cdot P(M)}{\int_{\bar{M}} P(x|\bar{M}) \cdot P(\mathbf{C}|\bar{M}) \cdot P(\bar{M})\ d\bar{M}}. \tag{B.16}$$

We compute $P(x|FG)$ and $P(x|BG)$ from color histograms whereas $P(FG) = \frac{|\Omega_f|}{|\Omega|}$ and $P(BG) = \frac{|\Omega_b|}{|\Omega|}$. While the marginalization over foreground and background is straight-forward, it is intractable for the model pose space. As mentioned in the paper, we assume both $P(x|M)$ and $P(M)$ to be uniform. Furthermore, since the integration over all valid M of our cloud term $P(\mathbf{C}|M)$ is constant, we reduce ourselves to a proportionate measure.

Bibliography

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. In *OSDI*.

[Aldoma et al., 2015] Aldoma, A., Tombari, F., Di Stefano, L., and Vincze, M. (2015). A Global Hypothesis Verification Framework for 3D Object Recognition in Clutter. *TPAMI*.

[Aldoma et al., 2013] Aldoma, A., Tombari, F., Prankl, J., Richtsfeld, A., Di Stefano, L., and Vincze, M. (2013). Multimodal cue integration through Hypotheses Verification for RGB-D object recognition and 6DOF pose estimation. In *ICRA*.

[Aubry et al., 2014] Aubry, M., Maturana, D., Efros, A., Russell, B., and Sivic, J. (2014). Seeing 3D chairs : exemplar part-based 2D-3D alignment using a large dataset of CAD models. In *CVPR*.

[Aytar and Zisserman, 2014] Aytar, Y. and Zisserman, A. (2014). Immediate , scalable object category detection. In *CVPR*.

[Bargteil et al., 2006] Bargteil, A. W., Goktekin, T. G., O'brien, J. F., and Strain, J. a. (2006). A semi-Lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*, 25(1):19–38.

[Bentley, 1975] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

[Bernardini and Rushmeier, 2002] Bernardini, F. and Rushmeier, H. (2002). The 3D Model Aquisition Pipeline. *Computer Graphics Forum*, 21.

[Besl and McKay, 1992] Besl, P. and McKay, N. (1992). A Method for Registration of 3-D Shapes. *TPAMI*.

[Bibby and Reid, 2008] Bibby, C. and Reid, I. (2008). Robust Real-Time Visual Tracking using Pixel-Wise Posteriors. In *ECCV*.

[Brachmann et al., 2014] Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). Learning 6D Object Pose Estimation using 3D Object Coordinates. In *ECCV*.

[Brachmann et al., 2016] Brachmann, E., Michel, F., Krull, A., Yang, M. Y., Gumhold, S., and Rother, C. (2016). Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image. In *CVPR*.

[Brox et al., 2010] Brox, T., Rosenhahn, B., Gall, J., and Cremers, D. (2010). Combined Region and Motion-Based 3D Tracking of Rigid and Articulated Objects. *TPAMI*.

[Bylow et al., 2013] Bylow, E., Sturm, J., Kerl, C., Kahl, F., and Cremers, D. (2013). Real-time camera tracking and 3d reconstruction using signed distance functions. In *RSS*.

[Cai et al., 2013] Cai, H., Werner, T., and Matas, J. (2013). Fast detection of multiple textureless 3-D objects. In *ICVS*.

[Calakli and Taubin, 2011] Calakli, F. and Taubin, G. (2011). SSD: Smooth Signed Distance Surface Reconstruction. *Computer Graphics Forum*, 30(7).

[Chan et al., 2006] Chan, T., Esedoglu, S., and Nikolova, M. (2006). Algorithms for finding global minimizers of image segmentation and denoising models. *SIAM Journal on Applied Mathematics*, 66:1632–1648.

[Chen et al., 2013] Chen, J., Bautembach, D., and Izadi, S. (2013). Scalable real-time volumetric surface reconstruction. *ACM TOG*, 32(4):1.

[Chen and Medioni, 1991] Chen, Y. and Medioni, G. (1991). Object Modeling by Registration of Multiple Range Images. In *ICRA*.

[Choi and Christensen, 2013] Choi, C. and Christensen, H. (2013). RGB-D Object Tracking: A Particle Filter Approach on GPU. In *IROS*.

[Curless and Levoy, 1996] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *SIGGRAPH*.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *CVPR*.

[Dambreville et al., 2010] Dambreville, S., Sandhu, R., Yezzi, A., and Tannenbaum, A. (2010). A Geometric Approach to Joint 2D Region-Based Segmentation and 3D Pose Estimation Using a 3D Shape Prior. *SIAM Journal on Imaging Sciences*.

[Damen and Bunnun, 2012] Damen, D. and Bunnun, P. (2012). Real-time learning and detection of 3D texture-less objects: a scalable approach. In *BMVC*.

[Dean et al., 2013] Dean, T., Ruzon, M., Segal, M., Shlens, J., Vijayanarasimhan, S., and Yagnik, J. (2013). Fast, Accurate Detection of 100,000 Object Classes on a Single Machine. In *CVPR*.

[Dimashova et al., 2013] Dimashova, M., Lysenkov, I., Rabaud, V., and Eruhimov, V. (2013). Tabletop Object Scanning with an RGB-D Sensor. *ICRA Workshop*.

[Drost and Ilic, 2015] Drost, B. and Ilic, S. (2015). Graph-based deformable 3d object matching. In *GCPR*.

[Drost et al., 2010] Drost, B., Ulrich, M., Navab, N., and Ilic, S. (2010). Model globally, match locally: efficient and robust 3d object recognition. In *CVPR*.

[Drummond and Cipolla, 2002] Drummond, T. and Cipolla, R. (2002). Real-time visual tracking of complex structures. *TPAMI*.

[Endres et al., 2012] Endres, F., Hess, J., Engelhard, N., Sturm, J., and Burgard, W. (2012). An Evaluation of the RGB-D SLAM System. In *ICRA*.

[Felzenszwalb et al., 2010] Felzenszwalb, P., Girshick, R., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part based models. In *TPAMI*.

[Felzenszwalb and Huttenlocher, 2012] Felzenszwalb, P. F. and Huttenlocher, D. P. (2012). Distance Transforms of Sampled Functions. *Theory of Computing*.

[Fidler et al., 2012] Fidler, S., Dickinson, S., and Urtasun, R. (2012). 3d object detection and viewpoint estimation with a deformable 3d cuboid model. In *NIPS*.

[Fitzgibbon, 2001] Fitzgibbon, A. (2001). Robust registration of 2D and 3D point sets. In *BMVC*.

[Fujiwara et al., 2011] Fujiwara, K., Nishino, K., Takamatsu, J., Zheng, B., and Ikeuchi, K. (2011). Locally rigid globally non-rigid surface registration. In *ICCV*.

[Fukushima, 1980] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.

[Gall et al., 2011] Gall, J., Yao, A., Razavi, N., Van Gool, L., and Lempitsky, V. (2011). Hough forests for object detection, tracking, and action recognition. *TPAMI*.

[Gionis et al., 1999] Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *VLDB*.

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., Berkeley, U. C., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

[Gong et al., 2013] Gong, Y., Lazebnik, S., Gordo, A., and Perronnin, F. (2013). Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*.

[Graber et al., 2011] Graber, G., Pock, T., and Bischof, H. (2011). Online 3D reconstruction using convex optimization. In *ICCV Workshop*.

[Gu and Ren, 2010] Gu, C. and Ren, X. (2010). Discriminative mixture-of-templates for viewpoint classification. In *ECCV*.

[Habert et al., 2015] Habert, S., Meng, M., Kehl, W., Wang, X., Tombari, F., Fallavollita, P., and Navab, N. (2015). Augmenting mobile C-arm fluoroscopes via Stereo-RGBD sensors for multimodal visualization. In *ISMAR*.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *BMVC*.

[Hartley and Zisserman, 2004] Hartley, R. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

[Hexner and Hagege, 2016] Hexner, J. and Hagege, R. R. (2016). 2D-3D Pose Estimation of Heterogeneous Objects Using a Region Based Approach. *IJCV*.

[Hinterstoisser et al., 2012a] Hinterstoisser, S., Cagniart, C., Ilic, S., Sturm, P., Navab, N., Fua, P., and Lepetit, V. (2012a). Gradient Response Maps for Real-Time Detection of Textureless Objects. *TPAMI*.

[Hinterstoisser et al., 2012b] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012b). Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *ACCV*.

[Hodan et al., 2016] Hodan, T., Matas, J., and Obdrzalek, S. (2016). On Evaluation of 6D Object Pose Estimation. In *ECCV Workshop*.

[Hodan et al., 2018] Hodan, T., Michel, F., Brachmann, E., Kehl, W., Buch, A., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., Sahin, C., Manhardt, F., Tombari, F., Kim, T.-K., Matas, J., and Rother, C. (2018). BOP: Benchmark for 6D Object Pose Estimation. In *ECCV*.

[Hodan et al., 2015] Hodan, T., Zabulis, X., Lourakis, M., Obdrzalek, S., and Matas, J. (2015). Detection and Fine 3D Pose Estimation of Textureless Objects in RGB-D Images. In *IROS*.

[Hoiem and Savarese, 2011] Hoiem, D. and Savarese, S. (2011). *Representations and techniques for 3D object recognition and scene interpretation*. Morgan & Claypool Publishers.

[Houston et al., 2006] Houston, B., Nielsen, M. B., Batty, C., Nilsson, O., and Museth, K. (2006). Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM TOG*, 25(1).

[Huber and Hebert, 2003] Huber, D. F. and Hebert, M. (2003). Fully automatic registration of multiple 3D data sets. *IVC*, 21.

[Innmann et al., 2016] Innmann, M., Zollhöfer, M., Nießner, M., Theobalt, C., and Stamminger, M. (2016). Volumedeform: Real-time volumetric non-rigid reconstruction. In *ECCV*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. In *ICML*.

[Jégou et al., 2011] Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor Search. *TPAMI*, 33(1).

[Jia et al., 2014] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. Technical report.

[Jones et al., 2006] Jones, M. W., Baerentzen, J. A., and Sramek, M. (2006). 3D distance fields: A survey of techniques and applications. *IEEE TVCG*, 12(4).

[Kehl et al., 2016a] Kehl, W., Holl, T., Tombari, F., Ilic, S., and Navab, N. (2016a). An Octree-Based Approach towards Efficient Variational Range Data Fusion. In *BMVC*.

[Kehl et al., 2017a] Kehl, W., Manhardt, F., Ilic, S., Tombari, F., and Navab, N. (2017a). SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. In *ICCV*.

[Kehl et al., 2016b] Kehl, W., Milletari, F., Tombari, F., Ilic, S., and Navab, N. (2016b). Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation. In *ECCV*.

[Kehl et al., 2014] Kehl, W., Navab, N., and Ilic, S. (2014). Coloured signed distance fields for full 3D object reconstruction. In *BMVC*.

[Kehl et al., 2017b] Kehl, W., Tombari, F., Ilic, S., and Navab, N. (2017b). Real-Time 3D Model Tracking in Color and Depth on a Single CPU Core. In *CVPR*.

[Kehl et al., 2015] Kehl, W., Tombari, F., Navab, N., Ilic, S., and Lepetit, V. (2015). Hashmod: A Hashing Method for Scalable 3D Object Detection. In *BMVC*.

[Kendall et al., 2015] Kendall, A., Grimes, M., and Cipolla, R. (2015). PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization. In *ICCV*.

[Kerl et al., 2013] Kerl, C., Sturm, J., and Cremers, D. (2013). Robust odometry estimation for RGB-D cameras. In *ICRA*. Ieee.

[Kingma and Ba, 2015] Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

[Klingensmith et al., 2015] Klingensmith, M., Dryanovski, I., Srinivasa, S., and Xiao, J. (2015). CHISEL: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially-Hashed Signed Distance Fields. In *RSS*.

[Krainin et al., 2011] Krainin, M., Henry, P., Ren, X., and Fox, D. (2011). Manipulator and object tracking for in-hand 3D object modeling. *IJRR*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

[Krull et al., 2014] Krull, A., Michel, F., Brachmann, E., Gumhold, S., Ihrke, S., and Rother, C. (2014). 6-DOF Model Based Tracking via Object Coordinate Regression. In *ACCV*.

[Kubacki et al., 2012] Kubacki, D. B., Bui, H. Q., Babacan, S., and Do, M. (2012). Registration and integration of multiple depth images using signed distance function. In *SPIE 8296, Computational Imaging X*.

[Kummerle et al., 2011] Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). G2o: A general framework for graph optimization.

[Lee et al., 2006] Lee, S.-I., Lee, H., Abbeel, P., and Ng, A. (2006). Efficient L1 Regularized Logistic Regression. In *AAAI*.

[Lepetit et al., 2008] Lepetit, V., Moreno-Noguer, F., and Fua, P. (2008). Epnp: An accurate o(n) solution to the pnp problem. In *TPAMI*.

[Li et al., 2018] Li, R., Wang, S., Long, Z., and Gu, D. (2018). UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning. In *ICRA*.

[Lin et al., 2014] Lin, G., Shen, C., Shi, Q., Hengel, A. V. D., and Suter, D. (2014). Fast Supervised Hashing with Decision Trees for High-Dimensional Data. In *CVPR*.

[Lin et al., 2017] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *ICCV*.

[Liu et al., 2016] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-y., and Berg, A. C. (2016). SSD : Single Shot MultiBox Detector. In *ECCV*.

[Lorensen and Cline, 1987] Lorensen, W. and Cline, H. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*.

[Losasso et al., 2006] Losasso, F., Fedkiw, R., and Osher, S. (2006). Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids*, 35(10).

[Losasso et al., 2004] Losasso, F., Gibou, F., and Fedkiw, R. (2004). Simulating water and smoke with an octree data structure. *ACM TOG*, 23(3).

[Lowe, 1992] Lowe, D. (1992). Fitting parameterized three-dimensional models to images. In *TPAMI*.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *IJCV*.

[Makadia et al., 2006] Makadia, A., Patterson, A., and Daniilidis, K. (2006). Fully Automatic Registration of 3D Point Clouds. In *CVPR*.

[Manhardt et al., 2019a] Manhardt, F., Arroyo, D., Rupprecht, C., Busam, B., Birdal, T., Navab, N., and Tombari, F. (2019a). Explaining the ambiguity of object detection and 6d pose from visual data. In *ICCV*.

[Manhardt et al., 2019b] Manhardt, F., Kehl, W., and Gaidon, A. (2019b). ROI-10D: Monocular Lifting of 2D Detection to 6D Pose and Metric Shape. In *CVPR*.

[Manhardt et al., 2018] Manhardt, F., Kehl, W., Navab, N., and Tombari, F. (2018). Deep Model-Based 6D Pose Refinement in RGB. In *ECCV*.

[Masci et al., 2011] Masci, J., Meier, U., Ciresan, D., and Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *ICANN*.

[Meagher, 1982] Meagher, D. (1982). Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(1):85.

[Milletari et al., 2015] Milletari, F., Kehl, W., Tombari, F., , Ilic, S., Ahmadi, A., and Navab, N. (2015). Universal Hough dictionaries for object tracking. In *BMVC*.

[Muja and Lowe, 2014] Muja, M. and Lowe, D. (2014). Scalable nearest neighbour methods for high dimensional data. *TPAMI*.

[Nayar et al., 1996] Nayar, S., Nene, S., and Murase, H. (1996). Real-time 100 object recognition system. In *ICRA*.

[Newcombe et al., 2015] Newcombe, R., Baust, D. F., and Seitz, S. (2015). Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time.

[Newcombe et al., 2011] Newcombe, R. A., Davison, A. J., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., Molyneaux, D., Hodges, S., Kim, D., and Fitzgibbon, A. (2011). KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*.

[Nießner et al., 2013] Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3D Reconstruction at Scale Using Voxel Hashing. *TOG*.

[Nissler et al., 2017] Nissler, C., Badshah, I., Castellini, C., Kehl, W., and Navab, N. (2017). Improving Optical Myography via Convolutional Neural Networks. In *MEC*.

[Nistér and Stewénius, 2006] Nistér, D. and Stewénius, H. (2006). Scalable Recognition with a Vocabulary Tree. In *CVPR*.

[Ochs et al., 2013] Ochs, P., Dosovitskiy, A., Brox, T., and Pock, T. (2013). An Iterated L1 Algorithm for Non-smooth Non-convex Optimization in Computer Vision. In *CVPR*.

[Paragios et al., 2002] Paragios, N., Rousson, M., and Ramesh, V. (2002). Matching distance functions: A shape-to-area variational approach for global-to-local registration. In *ECCV*.

[Pepik et al., 2012] Pepik, B., Gehler, P., Stark, M., and Schiele, B. (2012). 3D2Pm–3D Deformable Part Models. In *ECCV*.

[Popinet, 2003] Popinet, S. (2003). Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Computational Physics*, 190(20).

[Prisacariu et al., 2015] Prisacariu, V. A., Murray, D. W., and Reid, I. D. (2015). Real-Time 3D Tracking and Reconstruction on Mobile Phones. *TVCG*.

[Prisacariu and Reid, 2012] Prisacariu, V. A. and Reid, I. D. (2012). PWP3D: Real-Time Segmentation and Tracking of 3D Objects. *IJCV*.

[Rad and Lepetit, 2017] Rad, M. and Lepetit, V. (2017). BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. In *ICCV*.

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *CVPR*.

[Ren et al., 2014] Ren, C. Y., Prisacariu, V., Kaehler, O., Reid, I., and Murray, D. (2014). 3D Tracking of Multiple Objects with Identical Appearance using RGB-D Input. In *3DV*.

[Ren et al., 2013] Ren, C. Y., Prisacariu, V., Murray, D., and Reid, I. (2013). STAR3D: Simultaneous tracking and reconstruction of 3D objects using RGB-D data. In *ICCV*.

[Ren and Reid, 2012] Ren, C. Y. and Reid, I. (2012). A unified energy minimization framework for model fitting in depth. In *ECCV*.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.

[Rios-Cabrera and Tuytelaars, 2013] Rios-Cabrera, R. and Tuytelaars, T. (2013). Discriminatively trained templates for 3d object detection: A real time scalable approach. In *ICCV*.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method.

[Rosenblatt, 1956] Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a densitiy function. *The Annals of Mathematical Statistics*, 27:832–837.

[Rosenhahn et al., 2006] Rosenhahn, B., Brox, T., Cremers, D., and Seidel, H. P. (2006). A comparison of shape matching methods for contour based pose estimation. *LNCS*.

[Rouhani and Sappa, 2013] Rouhani, M. and Sappa, A. (2013). The Richer Representation the Better Registration. In *IEEE TIP*.

[Rusinkiewicz et al., 2002] Rusinkiewicz, S., Hall-Holt, O., and Levoy, M. (2002). Real-Time 3D Model Acquisition. In *SIGGRAPH*.

[Rusinkiewicz and Levoy, 2001] Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the ICP algorithm. In *3DIM*.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *IJCV*.

[Schmaltz et al., 2007] Schmaltz, C., Rosenhahn, B., Brox, T., Cremers, D., Weickert, J., Wietzke, L., and Sommer, G. (2007). Region-Based Pose Tracking. In *IbPRIA*.

[Schmaltz et al., 2012] Schmaltz, C., Rosenhahn, B., Brox, T., and Weickert, J. (2012). Region-based pose tracking with occlusions using 3D models. *MVA*.

[Schroers et al., 2012] Schroers, C., Zimmer, H., Valgaerts, L., Demetz, O., and Weickert, J. (2012). Anisotropic Range Image Integration. *Pattern Recognition, LNCS*.

[Seo et al., 2014] Seo, B. K., Park, H., Park, J. I., Hinterstoisser, S., and Ilic, S. (2014). Optimal local searching for fast and robust textureless 3D object tracking in highly cluttered backgrounds. In *TVCG*.

[Slavcheva et al., 2018] Slavcheva, M., Baust, M., and Ilic, S. (2018). Sobolevfusion: 3d reconstruction of scenes undergoing free non-rigid motion.

[Slavcheva et al., 2016] Slavcheva, M., Kehl, W., Navab, N., and Ilic, S. (2016). SDF-2-SDF: Highly Accurate 3D Object Reconstruction. *ECCV*.

[Slavcheva et al., 2017] Slavcheva, M., Kehl, W., Navab, N., and Ilic, S. (2017). SDF-2-SDF Registration for Real-Time 3D Reconstruction from RGB-D Data. *IJCV*.

[Steinbrucker et al., 2013] Steinbrucker, F., Kerl, C., Sturm, J., and Cremers, D. (2013). Large-scale multi-resolution surface reconstruction from RGB-D sequences. In *ICCV*.

[Steinbrucker et al., 2011] Steinbrucker, F., Sturm, J., and Cremers, D. (2011). Real-time visual odometry from dense RGB-D images. In *ICCV Workshop*.

[Strain, 1999] Strain, J. (1999). Tree Methods for Moving Interfaces. *Computational Physics*, 151(2).

[Szegedy et al., 2016] Szegedy, C., Ioffe, S., and Vanhoucke, V. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arxiv:1602.07261*.

[Tan and Ilic, 2014] Tan, D. J. and Ilic, S. (2014). Multi-forest tracker: A Chameleon in tracking. In *CVPR*.

[Tan et al., 2015] Tan, D. J., Tombari, F., Ilic, S., and Navab, N. (2015). A Versatile Learning-based 3D Temporal Tracker : Scalable , Robust , Online. In *ICCV*.

[Tateno et al., 2009] Tateno, K., Kotake, D., and Uchiyama, S. (2009). Model-based 3D Object Tracking with Online Texture Update. In *MVA*.

[Tateno et al., 2016] Tateno, K., Tombari, F., and Navab, N. (2016). When 2.5d is not enough: Simultaneous reconstruction, segmentation and recognition on dense slam.

[Tejani et al., 2014] Tejani, A., Tang, D., Kouskouridas, R., and Kim, T.-k. (2014). Latent-class hough forests for 3D object detection and pose estimation. In *ECCV*.

[Tjaden et al., 2016] Tjaden, H., Schwanecke, U., and Schoemer, E. (2016). Real-Time Monocular Segmentation and Pose Tracking of Multiple Objects. In *ECCV*.

[Tjaden et al., 2019] Tjaden, H., Schwanecke, U., Schömer, E., and Cremers, D. (2019). A region-based gauss-newton approach to real-time monocular multiple object tracking. In *TPAMI*.

[Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*.

[Tremblay et al., 2018] Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., and Dieter Fox, S. B. (2018). Deep object pose estimation for semantic robotic grasping of household objects. In *CoRL*.

[Ulrich et al., 2012] Ulrich, M., Wiedemann, C., and Steger, C. (2012). Combining scale-space and similarity-based aspect graphs for fast 3D object recognition. *TPAMI*.

[Weise et al., 2011] Weise, T., Wismer, T., Leibe, B., and Van Gool, L. (2011). Online loop closure for real-time interactive 3D scanning. *CVIU*.

[Whelan et al., 2013] Whelan, T., Johannsson, H., Kaess, M., Leonard, J. J., and McDonald, J. (2013). Robust real-time visual odometry for dense RGB-D mapping. In *ICRA*.

[Wohlhart and Lepetit, 2015] Wohlhart, P. and Lepetit, V. (2015). Learning Descriptors for Object Recognition and 3D Pose Estimation. In *CVPR*.

[Xiang et al., 2018] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2018). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes.

[Yuxin Wu, 2018] Yuxin Wu, K. H. (2018). Group normalization. In *ECCV*.

[Zach et al., 2008] Zach, C., Gallup, D., Frahm, J., and Niethammer, M. (2008). Fast Global Labeling for Real-Time Stereo Using Multiple Plane Sweeps. In *Proc. of Vision, Modeling and Visualization Workshop*.

[Zach et al., 2007] Zach, C., Pock, T., and Bischof, H. (2007). A globally optimal algorithm for robust TV-L1 range image integration. In *ICCV*.

[Zakharov et al., 2019] Zakharov, S., Kehl, W., and Ilic, S. (2019). DeceptionNet: Network-Driven Domain Randomization. In *ICCV*.

[Zakharov et al., 2017] Zakharov, S., Kehl, W., Planche, B., Hutter, A., and Ilic, S. (2017). 3D object instance recognition and pose estimation using triplet loss with dynamic margin. In *IROS*.

[Zeng et al., 2013] Zeng, M., Zhao, F., Zheng, J., and Liu, X. (2013). Octree-based fusion for realtime 3D reconstruction. *Graphical Models*, 75(3).

[Zhao et al., 2014] Zhao, S., Wang, L., Sui, W., Wu, H. Y., and Pan, C. (2014). 3D object tracking via boundary constrained region-based model. In *ICIP*.

[Zollhöfer et al., 2014] Zollhöfer, M., Nießner, M., Izadi, S., Rhemann, C., Zach, C., Fisher, M., Wu, C., Fitzgibbon, A., Loop, C., Theobalt, C., and Stamminger, M. (2014). Real-time non-rigid reconstruction using an rgb-d camera. In *TOG*.