Model-Based Self-Supervision for Fine-Grained Image Understanding

Tanner Schmidt

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

Doctor of Philosophy

UNIVERSITY OF WASHINGTON
2019

Reading Committee:

Dieter Fox, Chair
Ali Farhadi
Richard Newcombe

Program Authorized to Offer Degree:

Computer Science & Engineering

UNIVERSITY OF WASHINGTON

Abstract

Model-Based Self-Supervision for Fine-Grained Image Understanding

Tanner Schmidt

Chair of the Supervisory Committee:

Dieter Fox
Computer Science & Engineering

In order to enable robots that perform tasks capably, efficiently, and safely in dynamic environments, we'll need vision systems that are adaptable and can reliably provide detailed and accurate information about the world around the robot. Generative model-based methods are well suited to these demands, as generative models are portable (they are equally valid in many different environments), modular (they can be dynamically combined with other models), and interpretable (via likelihood functions and explicitly parameterized state spaces). For robotics applications, these properties give model-based vision an advantage over the currently dominant deep learning paradigm in computer vision. However, generative models are limited by the curse of dimensionality and data association ambiguity, and are thus plagued by a lack of robustness which is an impediment to successful deployment in robotic systems.

This dissertation advances the argument that the strengths and weaknesses of generative model-based vision and discriminative learning-based vision are largely complementary, and can therefore be used to mutual advantage. Specifically, we'll show that generative models can be employed to automate the process of labeling data for supervised training of deep neural networks, demonstrating the benefit to deep learning offered by model-based vision. Then, we'll show that deep neural networks trained to recognize parts of generative models can be used to help resolve ambiguous data association and thus enhance the robustness of state estimation, demonstrating the benefits deep learning can bring back to model-based vision. Overall, we lay out a vision for future development of highly capable robot perception systems in which machine learning expands the envelope of situations in which generative model-based techniques are reliably applicable, and the generative model-based techniques return the favor by providing labels for further training of the network.

"Everything that falls upon the eye is apparition, a sheet dropped over the world's true workings."

Marilynne Robinson, "Housekeeping"

Acknowledgements

First, I would like to thank Dieter Fox for his support, advice, and encouragement throughout my Ph.D. studies. His insight and incredibly broad and deep knowledge have been invaluable. His ability to keep his finger on the pulse of both computer vision and robotics research is uncanny, and it was my privilege to work with him during my years at the University of Washington.

I would also like to thank Richard Newcombe for his mentorship, technical and otherwise, during the critical first years of my Ph.D. studies; if he had not been so generous with his time, it is hard to imagine I would be where I am today.

Thanks to all of my labmates in the Robotics and State Estimation lab over the years for the laughs, commiseration, and great discussions. Many thanks to all the educators who have graciously helped me on the long journey to this point, and especially to Jeffrey Forbes and Ron Parr for getting me started on research as an undergraduate and sparking my enthusiasm for artificial intelligence, robotics, and computer vision.

Finally, I would like to thank my family and friends for all the love and support through the long years of late nights. Mostly, thank you, Claire — I couldn't have done this without you.

Contents

Chapter 1

Introduction

In recent years, great strides have been made in the field of computer vision, due in large part to the availability of massively parallel computing hardware that enables the relatively efficient training of deep artificial neural networks. The AlexNet model by Krizhevsky, Sutskever, and Hinton, (2012) is often credited with inspiring the recent surge of literature on the topic of deep learning for computer vision by showing that a deep convolutional neural network trained on the ImageNet dataset (Deng et al., 2009) could achieve a (top-5) classification accuracy of 83.0%. Within three years, incremental improvements in network architectures and training procedures brought this figure up to 96.43%, achieved using a network with 152 layers (He et al., 2016).

Meanwhile, these and other techniques led to breakthroughs in other visual tasks, such as semantic segmentation (Long, Shelhamer, and Darrell, 2015), object detection and recognition (Girshick et al., 2014; He et al., 2017), and human pose estimation (Wei et al., 2016). As these results and others have galvanized the computer vision community, they have also generated considerable excitement within the robotics community, which continues to grapple with the problem of robot perception. However, progress in vision has arguably not translated easily into success for robotics. While there have been exciting applications of deep learning to robot perception problems, the results have been somewhat less compelling.

As graphics programming units (GPUs) were accelerating learning-based vision systems, they also made a number of compute-intensive but highly parallel generative model-based, real-time vision algorithms tractable for the first time. A notable example is KinectFusion (Newcombe et al., 2011), which combined newly developed depth sensors and modern GPUs to build a dense scene model and track it in real time. These techniques are powerful, and because of the underlying model highly detailed information can be extracted from the data (in the case of KinectFusion, precise camera poses and metric maps accurate to at least the centimeter level). This fine-grained information has enabled a number of robotics applications. However, many real-time generative model-based vision systems suffer from a lack of robustness, failing to recover when tracking is lost. Even if tracking is maintained, as soon as the camera feed ends the model is of limited use to future observations of the same object if it cannot be reliably relocalized. This has the effect of relegating robots that depend on detailed information about their environment to the research lab where they will not get lost or hurt anyone.

This dissertation begins with an exploration of what differentiates more 'mainstream' approaches to computer vision from more robotics-focused approaches, and a hypothesis about why the successes on view at top computer vision conferences have yet to fully spill over into the robotics domain. It will then develop a potential solution

to this problem, based on the observation that model-based and learning-based approaches are in many ways complementary. Leveraging the strengths of both model-based and learning-based vision could lead to powerful new vision systems that enable widespread robotics applications.

1.1 Web-Scale Vision vs. Embodied Vision

For the remainder of this dissertation I will refer to the 'mainstream' approaches to computer vision as 'web-scale' vision. This term nicely encapsulates the ambitions and intended application domain for this line as work. In short, the goal of web-scale vision is to be able to understand any image that is likely to be posted somewhere on the web, either publicly or privately, at a level that enables applications that a typical web user might find useful. Such applications might include automatically tagging a friend or landmark in an image, geo-tagging a photo album, identifying interesting or salient objects in a photo, or categorizing large photo collections. In contrast, the goal of embodied vision is to understand a stream of images from a sensor well enough to interact with the local environment. Though there is significant overlap, achieving success in embodied vision may require a significantly different approach. To understand why, consider the differing demands on web-scale and embodied vision systems.

1.1.1 Web-Scale Vision

Perhaps the first and foremost property of web-scale vision is the scope: a web-scale vision application should run globally. In other words, it should be able to process any image of anything from anywhere in the world with reasonable results. It is impossible or at least impractical to collect a dataset covering any image that could possibly be taken in the future, so web-scale vision systems place a heavy emphasis on generalization performance.

FIGURE 1.1: An example output frame from Mask-RCNN. Reproduced from He et al., (2017).

On the other hand, along with the broad scope comes a generally high tolerance for error. For example, an automatic tagging system with an accuracy of 90% would likely be considered good enough for deployment today.

Current web-scale vision systems also tend towards coarseness in both a spatial and semantic sense. A web-scale application for object detection might be happy with localizing objects within a bounding box, and might be less likely to provide precise pixel-level segmentations of the object and even less likely to estimate poses.

For a typical web-scale vision system, the background of an image is often treated as a nuisance factor that the system must learn to ignore. Thus, while the broad scope demands invariance to backgrounds, the focus is on the foreground.

Finally, a web-scale vision system aims to operate reliably on any image drawn from the distribution of all images on the web. This distribution is in fact not as dense as it might seem at first blush. Most images on the web are composed, i.e. they are taken for an aesthetic or informational purpose and therefore often contain a clear subject.

This means that a rather sparse subspace of all possible images accounts for most of the probability density in the distribution of images on the web. In other words, a person is much more likely to take a picture of the Eiffel tower than of any of the trash cans they passed on the way there.

To ground this discussion somewhat, Figure 1.1 shows a sample of the output of the Mask-RCNN system by He et al., (2017), which is currently considered a state-of-the-art web-scale vision system. The results are undeniably impressive. Note that multiple instances of cars that are occluding each other are accurately segmented. Note also, however, how much is missing — well over half the image, including all the pixels associated with the ground and with buildings in the scene are unlabeled.

1.1.2 Embodied Vision

The demands of embodied vision differ on all these counts. Firstly, the scope of an embodied vision system is much more limited, encompassing only so much of the local physical environment as the embodied agent might traverse. Even if the size of this environment is large, there are also temporal constraints on the data. Each image arrives just a fraction of a second after the last, and the difference between the two is thus limited by the dynamics of the agent and of the environment. Generalization thus takes on a very different meaning. If the system can continue learning online without supervision, incoming frames immediately become part of the training set. Then, the 'test set' (i.e. the current image) is quite close to a number of images in the training set. Furthermore, if the embodied agent knows its geographical location and the current time of day, those values will not change significantly within a short time frame and therefore that knowledge can be used during inference about incoming observations.

While the reduced scope compared to web-scale vision makes the embodied vision problem easier, with it generally comes a lower tolerance for error. A robot operating heavy machinery in an environment shared with humans must obviously be extremely confident about the location and pose of all the nearby people in order to make satisfactory safety guarantees.

In addition to requiring higher accuracy, embodied vision typically demands much finer granularity than web-scale vision. To return to the above example, the ability to place a bounding box around the humans in an image of the scene will not suffice. Potentially, knowledge of the exact extent in three dimensions of the humans' body and clothing may be required.

In an embodied setting, pixels that a web-scale system would consider background may in fact be relevant for the task at hand. A web-scale system may consider floors uninteresting and forgo labeling them, but a robot seeking to place a foot should certainly be able to accurately perceive the floor. While a web-scale system might only provide information about a finite set of salient object categories, a mobile robot needs to avoid obstacles even if they are otherwise uninteresting or of an unknown class. Thus, background for embodied vision is not simply a nuisance factor but a source of potentially critical information.

Finally, images processed by an embodied vision system are in general captured for functional rather than intrinsic or aesthetic value, and therefore come from a much different distribution. To return to the Eiffel tower example, a mobile robot needs to be

able to understand all of the intermediate images of sidewalks, pedestrians, signs, cars, trees, and yes, even trash cans, in order to successfully navigate to the destination.

1.1.3 Convergent Goals?

One question which naturally arises at this point in the discussion is: don't web-scale and embodied vision have the same ultimate goal? Stated differently, couldn't we develop one superhuman vision system which satisfies the demands of both web-scale and embodied vision? While this would of course be ideal, in practice this will take a long time and a lot of work.

In the meantime, web-scale and embodied vision applications will inform different search strategies in the space of possible vision systems. We might think of web-scale vision research as a breadth-first search for ever more powerful vision systems, while embodied vision research is a depth-first search. An internet search company might prioritize understanding a little bit about the appearance of nearly everything, but it would be much more useful in the short term for a robot to understand nearly everything about the appearance of just a few things. While ultimately the goals of both approaches is to understand everything about the appearance of everything, roboticists cannot afford to wait until that is possible. This document will therefore propose work towards enabling robots to robustly and accurately perceive the state of their local environments.

1.2 Model-Based vs. Learning-Based

Before continuing the discussion we'll draw one additional distinction, this time between model-based and learning-based vision. These are broad categories and not every vision system fits nicely into one or the other, but we'll define a model-based vision approach to be any approach that involves an explicit parametric model which is fit to observed data by maximizing some likelihood function. A learning-based system, on the other hand, is a vision system which involves learning a function from data.

1.2.1 Generative Model-Based Vision

Model-based vision is perhaps a more natural way to approach the vision problem from an embodied perspective. We've already touched on one reason for this, which is the granularity of information provided by model-based vision. A robot seeking to grasp an object will likely be more successful when fitting an explicit 3D model of the object to observed data than when estimating its pose using a deep network.

There are other advantages to model-based vision. One is that it is **interpretable**, by virtue of the fact that the entire state of the system can be described by the parameters of the model. Furthermore, because state is estimated using a likelihood model, the certainty of the estimate is known and has a clear probabilistic interpretation according to the model. Model-based vision is also **portable**. If a model of an object is available, the model will be equally valid if the object is taken to a new environment with different illumination and / or surroundings. Finally, model-based vision is **modular**. If you have a model of a robot hand and a model of an object and each can be tracked in video, it should be easy to combine the models and track both the hand and object as they interact (we'll see this exact example in Chapter 3).

The key disadvantage of model-based vision is that as models become more complex and likelihood functions become more sophisticated, at some point it is no longer tractable to do inference in real time even with modern hardware and optimized code. This often leads to approximations, such as assuming that observations at every pixel location in an image are independent. These approximations work well, but tend to result in a narrowing of the basin of convergence (c.f. Chapter 5 for an exploration of this idea). This is fine as long as frame rates are high and an accurate state estimate can be maintained, but as soon as the state estimate diverges from the true state it can be very difficult to recover. Here, the curse of dimensionality poses an additional challenge for complex models, as many model-based reinitialization techniques such as RANSAC (Fischler and Bolles, 1981) become exponentially more expensive as the dimensionality of the model's state space increases.

1.2.2 Discriminative Learning-Based Vision

One potential way to avoid this problem is to take a learning-based approach. If enough observation-state pairs are available, one can train a model (e.g. a deep neural network) to map directly from observations to state estimates. Unlike many generative model-based techniques, this requires no prior on the state. Therefore, if the estimate of the state diverges from the true state, perhaps because of a series of highly ambiguous observations, the system will not be any less likely to estimate the correct state for the next observation.

As usual, there is no free lunch. To get the benefits of discriminative learning-based, one must sacrifice many of the advantages of model-based vision. Many learning-based models such as deep neural networks are notorious for being black boxes, i.e. it is incredibly difficult to interpret what factors led the network to produce a particular output. Due to the reliance on training data, learning-based systems are not by nature portable. If a system is trained in lighting conditions that differ significantly from the lighting conditions at the time of testing, the results will likely suffer. Thus, porting a learning-based system to a novel environment will generally require re-training, and training a system that works in *any* environment would require a massive training set and careful balancing. Finally, many learning-based models are not modular. For example, it is typically not the case that one can splice and recombine pieces of trained neural networks without retraining. To return to the previous example, if one network is trained to recognize parts of a robot hand and another is trained to recognize parts of an object, it is not possible (without re-training) to combine them into a single model that recognizes parts of both hand and object. In fact, even if the two were used independently, it's entirely possible that the networks would fail to recognize both hand and object when the object is held in the hand and the two occlude each other, especially if the networks were not trained with images of such a configuration.

1.3 Why is Fine-Grained Visual Understanding Hard?

Arguably the hardest and most universal computer vision problem is the correspondence problem: given a visual observation, i.e. a photon count on a particular pixel, what can be said about the surfaces and/or light sources that led to the observation, in

relation to another observation from a different time or to some known model? This problem crops up repeatedly in many visions problems, such as:

- **Feature Matching**: This is the canonical correspondence problem: given a set of feature detections in two (or more) images, find the corresponding detections.
- **Template Matching and Tracking**: These techniques rely on estimating an (often implicit) correspondence between observations and a known model or template. Typically, this involves alternating between estimating correspondence and optimizing template parameters assuming that the correspondence is correct.
- **Loop Closure**: This is a common problem in simultaneous localization and mapping (SLAM). Given a newly observed frame and a model or set of keyframes, correspondences between a new frame and previous observations are used to both detect loops and to close them.
- **Depth from Stereo**: Here, estimating correspondence between observations in the left and right images is what enables depth estimation by triangulation.
- **Optical Flow**: Optical flow is simply an estimation of the dense correspondence between pixels in a pair of images. Unlike depth from stereo, where the images are captured simultaneously, a nonzero amount of time elapses between capture and factors other than geometry and viewpoint can factor into the flow.
- **Object Detection, Recognition, and Segmentation**: This closely related set of vision problems can also be viewed as a set of correspondence problems. Recognizing an object involves identifying correspondence between an observation and a known model or class. Similarly, segmenting an object in an image is nothing more than estimating which pixels in the image are associated with the object.

Correspondence is extremely challenging to estimate in general, especially for dense, fine-grained tasks like optical flow and segmentation. One problem is that the appearance of a surface can change drastically with viewpoint and lighting conditions. Furthermore, in many cases there are actually fundamental ambiguities that prevent the full resolution of any true correspondence. The classical example is the aperture problem, but ambiguity also arises when surfaces are untextured such that multiple, equally reasonable correspondence hypotheses are consistent with all observations.

1.4 Dissertation Overview

This dissertation will explore that the idea that generative model-based techniques and discriminative learning-based techniques have complementary strengths, and that leveraging the strengths of both is a promising path forward for embodied vision. Learning-based approaches are often limited by the availability of high-quality training data; model-based techniques, in regimes in which they do work, can extract highly useful information from visual observations which can be used to self-supervise learning, obviating the need for large manually-labeled training sets. When model-based vision fails, it's often due to incorrect data association; a learning-based technique can be used

to learn features that aid in long-range correspondence estimation which, when incorporated into the model-based system, increases robustness and extends the operable range. Ultimately, this could lead to a virtuous cycle in which a model-based system labels data which is used to improve performance of a learning-based system, which is used to improve performance of the model-based system and increase the label extent and quality.

The dissertation will begin with model-based vision. Chapter 2 introduces a system for real-time tracking of arbitrary articulated models. Then, Chapter 3 shows how this system can be effectively applied to a robotics problem, in the process highlighting many of the aforementioned benefits of model-based vision for embodied vision. Chapter 4 will then show how this particular model-based system can be used to automate the data labeling process, allowing for deep networks to be trained to estimate object poses and / or semantic segmentations. Next, Chapter 5 will take a deep dive into the data association problem and how it affects the system described in Chapter 2 as well as generative model-based real-time reconstruction techniques. Chapter 6 will then show how these reconstruction techniques can also be used to automate the labeling process, in this case with the goal of training networks to aid in long-range correspondence estimation. This chapter will also show how the trained network can enable model re-initialization, providing a proof of concept for the virtuous cycle described above. Chapter 7 applies the approach of Chapter 6 to a larger-scale problem in which multiple objects are present simultaneously and moving independently. Chapter 8 will conclude with a discussion and future outlook for this line of work.

Chapter 2

Tracking Articulated Objects with a Generative Model

The ability to accurately track the pose of objects in real time is of fundamental importance to embodied vision. Applications of real-time object tracking range from navigation to planning, manipulation, and human-robot interaction, all of which have received the attention of researchers working within a generative model-based paradigm within both computer vision and robotics. The class of objects that can be described as collections of rigid bodies chained together through a kinematic tree is quite broad, including furniture, tools, human bodies, human hands, and robot manipulators. Tracking articulated bodies from a single viewpoint and without instrumenting the object of interest still presents a significant challenge where the single viewpoint and occlusions, including self-occlusion, limit the amount of information available for pose estimation. Noisy sensor data and approximate object models pose additional problems. Finally, the objects being tracked can be highly dynamic and have many degrees of freedom, making real-time tracking difficult.

Early articulated model-based tracking techniques relied on tracking 2D features such as image edges on a CPU (Drummond and Cipolla, 2001; Comport, Marchand, and Chaumette, 2007). As discussed in the Chapter 1, recently introduced depth cameras along with highly parallel algorithms optimized for modern GPUs have enabled new algorithms for tracking complex 3D objects in real time. In addition to KinectFusion (Newcombe et al., 2011), there have been other related 3D reconstruction efforts (Henry et al., 2013; Whelan et al., 2012), and work on human body pose tracking (Shotton et al., 2011; Ye and Yang, 2014; Helten et al., 2013) and articulated hand tracking (Oikonomidis, Kyriazis, and Argyros, 2011; Kyriazis and Argyros, 2013; Qian et al., 2014). These approaches were developed for specific application domains and have not been demonstrated or tested on multiple tracking applications. The application-specific nature of these approaches enables their authors to show excellent performance by taking advantage of domain-specific features and constraints, but it also prevents them from serving as general tools for tracking arbitrary articulated objects. Techniques have also been developed to track fully non-rigid deformations of an underlying surface template for both specific (Li et al., 2013) and general (Haehnel, Thrun, and Burgard, 2003; Li, Sumner, and Pauly, 2008; Schulman et al., 2013; Zollhöfer et al., 2014) object cases. However, the full generality of these models comes at the cost of increased model complexity, and for many objects which are well modeled as piecewise rigid bodies, such overparameterized output obscures the utility of tracking the articulated body state directly.

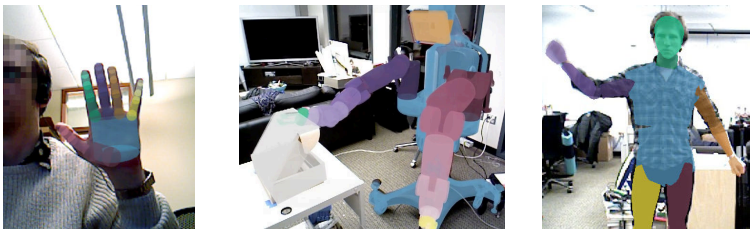


FIGURE 2.1: Articulated objects successfully tracked at frame-rate using DART, without object-specific tuning of the algorithm: a human hand, Rethink Robotics Baxter robot opening a box, and a full human body.

In this chapter we present Dense Articulated Real-time Tracking (DART), a general framework for tracking articulated models with formally defined kinematic and geometric structure using commodity depth sensors. DART represents objects with an articulated formulation of the signed distance function (SDF), which has been used to achieve very robust and efficient results for online 3D mapping (Newcombe et al., 2011; Henry et al., 2013; Whelan et al., 2012) and for tracking rigid objects in six degrees of freedom (Fitzgibbon, 2003; Ren and Reid, 2012; Newcombe, 2014; Canelhas, Stoyanov, and Lilienthal, 2013; Bylow, Olsson, and Kahl, 2014). Our tracking framework uses a local gradient-based approach to find the pose of the model which best explains the data points observed in a depth frame, starting from the pose estimate from the previous frame or, in the first frame, some initialization. The minimized objective function is trivially parallelizable, and optimized on a GPU, which allows us to use all available data to achieve accurate tracking results in real-time, even with models having as many as 48 degrees of freedom. To use DART in a new domain, one need only supply a model file specifying the relative locations of all the frames of reference, the possible articulations of the frames, and the geometry attached to each frame. Given this model, the object can be tracked with no changes to the code. DART is thus highly portable in the sense discussed in Chapter 1. The models can be designed manually, generated from a currently available volumetric fusion method, or derived from CAD or robot description models.

The main contributions of this DART are (1) a general framework for highly efficient tracking of articulated objects, (2) a generalization of signed distance functions to articulated objects, (3) a symmetric formulation of point set registration that incorporates negative information into the energy function, and (4) a demonstration of the framework tracking a variety of different models.

2.1 Articulated Tracking Overview

We can divide previous approaches to articulated model tracking into generative model-based techniques and discriminative learning-based techniques, and begin by looking at the former. Model-based techniques can further be divided into gradient-based methods and sampling based methods.

2.1.1 Gradient-based Generative Approaches

Methods in this category (which includes DART), involve the minimization of an objective function which is characterized by having partial derivatives with respect to the pose vector. These approaches are therefore able to take advantage of well-studied optimization techniques that make use of derivative information to incrementally improve an estimate of the optimum. Implicit surfaces and gradient descent on ICP-like error terms have proven successful in tracking rigid bodies, as by Newcombe et al., (2011), Sturm et al., (2013), and Henry et al., (2013). Implicit surfaces have also been used for model-based tracking of articulated models; Dewaele, Devernay, and Horaud, (2004) used an implicitly defined model and an expectation maximization approach to track human hands. Grest, Woetzel, and Koch, (2005) presented early work on human body tracking from depth, but subsampled the points and still did not reach real-time.

A human body tracking approach similar to DART is that of Ganapathi et al., (2012), which uses range data to track articulated motion of human bodies by augmenting traditional ICP with free space information. The work presented in this chapter differs by incorporating information about observed free space directly into the energy function, rather than using constraint projection. We also demonstrate the applicability of our approach to a wider set of application domains and demonstrate how our parallelizable optimization can take advantage of GPU acceleration.

More recently, Schulman et al., (2013) and Ye and Yang, (2014) used soft data associations between control points on a generative model and points observed from a depth camera along with an expectation-maximization (EM) optimization to find the pose. The approach of Ye and Yang, (2014) uses a rigged mesh model such that the control points are the vertices of the mesh and are deformed according to a linear combination of the transforms of the bones of the mesh skeleton. The focus of Schulman et al., (2013), on the other hand, was on fully deformable objects; their models are therefore mass-spring systems in which each control point moves freely, with some regularization over the distances between connected points. Both of these methods, as well as the work presented in this chapter, show that with the rich shape information provided by consumer depth cameras, a relatively simple objective function and optimization algorithm can be used to recover articulated poses. The main points of differentiation between our work and theirs is our treatment of negative information, and the density of our optimization. Both of these EM style approaches require a relatively sparse set of model points as well as a subsampling of the observed point cloud in order to reach real-time (or near real-time) performance, due to the $O(MN)$ complexity of the EM-ICP algorithm, where M is the number of model points and N is the number of observed points. In contrast, our precomputed signed distance functions allow us to store our surface models as the mathematical equivalent of an infinitely dense point cloud, and furthermore allows us to avoid subsampling the observed point cloud. This model density is particularly useful when tracking robots and other objects for which we have an exact model, as we can leverage high-frequency geometric features of the model to constrain the pose without sacrificing tracking speed.

Schröder et al., 2013 use a very similar articulated ICP formulation, called inverse kinematics hand tracking, following the observation that once data association has been determined, the resulting optimization is exactly the familiar (to the robotics community) inverse kinematics problem. The interesting innovation in this work is the use of

'synergies' or lower-dimensional representations of hand poses to track more efficiently and to infer values of pose parameters for which no observation is available. This addition is orthogonal to our method and could be incorporated into DART, although it would require a model-specific (and perhaps even task-specific) training set.

Articulated ICP error terms minimized by gradient descent have also been explored in the context of robot manipulators. Krainin et al., 2011 used an articulated ICP error term with a Kalman filter to track an arm-object system. This resulted in end effector estimates which were accurate enough to build a 3D model of the in-hand object. Likewise, Klingensmith et al., 2013 used an articulated ICP error to do visual servoing of a robot manipulator towards a point.

2.1.2 Sampling-based Generative Approaches

Unlike gradient-based approaches, sampling-based approaches rely on objective functions that are cheap to evaluate, but have partial derivatives which are either too expensive to compute or which do not give reliable information about the direction of the true minimum. For example, many articulated model tracking approaches in this category utilize some form of silhouette information, necessitating the use of indicator functions which are discontinuous and therefore not globally differentiable. Such methods therefore often rely on inference techniques which are less thoroughly studied and perhaps less theoretically justified, such as particle swarm optimization (PSO) as employed by Oikonomidis, Kyriazis, and Argyros, (2011) and Kyriazis and Argyros, (2013).

2.1.3 Discrimative Approaches

Finally, discriminative or appearance-based methods utilize training sets of observation-label pairs and machine learning techniques to learn a direct map from input images to specific articulated part labels and poses. Recently, random forests in particular have proven extremely successful for the problem of human pose estimation (Shotton et al., 2011; Taylor et al., 2012), and similar approaches have also been applied to hand tracking (Keskin et al., 2013; Kirac, Kara, and Akarun, 2014) and localizing and tracking objects in six degrees of freedom (Brachmann et al., 2014). Romero et al., (2013) use a nearest neighbor look-up in a large database of images of hands, including hands interacting with objects, to do real-time pose estimation. Tompson et al., (2014) use convolutional neural networks to track human hands from depth images. While these methods can be highly efficient and robust, they require a lot of data and time to train recognition models for each new object. In any case, these approaches could also be seen as complementary to ours, as they are able to provide pose estimates with little or no prior and could be used to increase tracking robustness in ways which will be further explored in later chapters. Chang and Zwicker, (2008) presented compelling work on a discriminative articulated alignment technique by matching shape features between point clouds which does not require any training data or even a template. However, their results are shown on very high quality and high-resolution point clouds such that reliable shape matching is possible, while we generally work with much noisier sensor data.

A combination of discriminative and generative approaches has been applied to articulated tracking, e.g. by Ballan et al., (2012) and Sridhar, Oulasvirta, and Theobalt, (2013), who used fingernail detectors to help constrain a generative model approach,

by Ganapathi et al., (2010), who used a gradient-based optimizer in conjunction with body part recognition for human body tracking, by Pauwels et al., (2014), who used a combination of articulated ICP and visual features to do visual servoing with RGB-D data, and by Qian et al., (2014), who combine particle swarm optimization, a standard ICP formulation, and a manually-designed finger detector to track human hands, thus straddling all three categories.

While many of these existing techniques achieve excellent results in their application domain, such as hand/object tracking (Oikonomidis, Kyriazis, and Argyros, 2011; Kyriazis and Argyros, 2013; Qian et al., 2014), human pose tracking (Shotton et al., 2011; Taylor et al., 2012; Ganapathi et al., 2010; Ganapathi et al., 2012; Ye and Yang, 2014), or deformable object tracking (Schulman et al., 2013), none of them have been demonstrated to work in several domains. In contrast, our DART framework relies on a highly efficient representation and optimization to operate under very general conditions, thereby enabling it to track a wide variety of objects.

2.2 DART: Dense Articulated Model Tracking

This section gives technical details on how DART works, including the model representation, the objective function, and how it is optimized.

2.2.1 Model Representation

The models we track are defined as a set of interconnected rigid bodies arranged in a kinematic tree. By convention, we designate the root of the tree as frame of reference 0. Every other frame of reference in the kinematic tree is attached to some other frame which is designated as its parent. The frame of reference i (for $i \neq 0$) is defined relative to its parent via the transform $T_{parent(i),i}(\theta) \in \mathbb{SE}(3)$, which may be a function of any subset of joint parameters θ (which collectively describe the articulated pose of the entire model). The transformation from frame i to any other frame j is then defined recursively according to

$$T_{j,i}(\theta) = T_{j,parent(i)}(\theta) * T_{parent(i),i}(\theta) \,, \tag{2.1}$$

such that the transform between any two frames is given by composing the transforms in a chain indexed using the $parent(\cdot)$ function. This recursive definition is the reason we assume a tree structure for all tracked models, which guarantees that there is a unique path between any two frames of reference. In the case where i and j are on different branches of the kinematic tree, the relative transform is formed by recursively chaining transforms up the tree from j to the model root, and then chaining transforms back down the tree from the root to i. Figure 2.2 shows a simple example of a kinematic tree.

Each frame may (but is not required to) include geometric elements, which are assumed to be rigidly affixed within the local frame of reference. Geometry attached to frame i is defined implicitly by a signed distance function $\text{SDF}^i(x) : \mathbb{R}^3 \rightarrow \mathbb{R}$, which takes on negative values inside the geometry, positive values outside, and therefore has a value of zero at the surface interface. In other words, the geometry is defined to be the zero-level set of the signed distance function. While the SDF is in principle defined for all $x \in \mathbb{R}^3$, discretizations of signed distance functions are, in practice, sampled

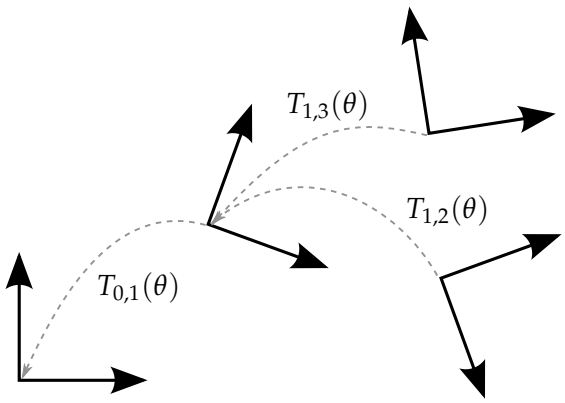


FIGURE 2.2: A 2D illustration of a kinematic tree. The pose of cordinate frames 2 and 3 relative to the root frame is defined recursively through coordinate frame 1, according to the pose parameter vector θ.

within finite bounds derived from application-specific requirements and practical computational and memory restrictions[1] Examples of typical voxelization bounds can be seen in Figure 2.4(A). In our experiments, these functions are either generated from an analytical function describing primitive shapes such as cylinders and ellipsoids, or by voxelizing closed polygonal meshes followed by computing a sampled Euclidean SDF using the efficient 3D distance transform algorithm of Felzenszwalb and Huttenlocher, (2012). Alternatively, the SDF functions could easily be generated from real-world objects for which no model is readily available using an efficient 3D scanning method, such as those developed by Newcombe et al., (2011), Sturm et al., (2013), or Henry et al., (2013). This could even be done autonomously with a robot arm, as shown by Krainin et al., (2011).

More formally, our model can be denoted as

$$\mathcal{M} = \left\{ \left(T_{parent(i),i}(\theta),\ \mathrm{SDF}^i(\mathbf{x}) \right)_{i=0:M-1} \right\}, \tag{2.2}$$

a set of M pairs of signed distance functions (in the local frame of reference) and mappings from the local coordinate frame to the parent coordinate frame as a function of the pose parameters.

2.2.2 Measurement Model and Data Association

We work with an input device capable of producing a depth map $D(\mathbf{u})$, which maps a pixel $\mathbf{u} = (u, v) \in \Omega$ to a depth value $d \in \mathbb{R}$, where $\Omega \subset \mathbb{R}^2$ is the image plane. We assume that these depth frames are generated probabilistically from some distribution $p(D|\theta; \mathcal{M})$, where $\theta \in \mathbb{R}^N$ is the vector describing the pose of the model and N is the

[1] Assuming an object is successfully tracked, observed points should lie near the model surface according to the pose estimate. When a new frame arrives, the distance between points and the model surface is then limited by the maximum velocity of any point. The SDF can therefore be sized by fitting the tightest possible bounding box around the geometry and then extending it in all directions according to a conservative estimate of the maximum motion from frame to frame.

number of degrees of freedom in the model. For the remainder of this section we will omit $\mathcal{M}$ for clarity.

The goal of pose estimation is to determine the pose vector θ which maximizes the likelihood of an observed depth map. To make this task more tractable, we make the common assumption that the probability of generating the depth value at each pixel $\mathbf{u}$ of the depth map is independent of the value at all other pixels, given the pose vector. This allows us to express the likelihood as

$$p(D|\theta) = \prod_{\mathbf{u}\in\Omega} p(D(\mathbf{u})|\theta) \,. \tag{2.3}$$

In theory, the depth observation at each pixel is the true depth $\hat{D}(\mathbf{u})$ from the camera to the model along the optical axis, corrupted by Gaussian noise. This defines the per-pixel depth measurement likelihood as

$$p(D(\mathbf{u})|\hat{D}(\mathbf{u})) \propto \exp(-(e(\mathbf{u})^2/\sigma^2) \,, \tag{2.4}$$

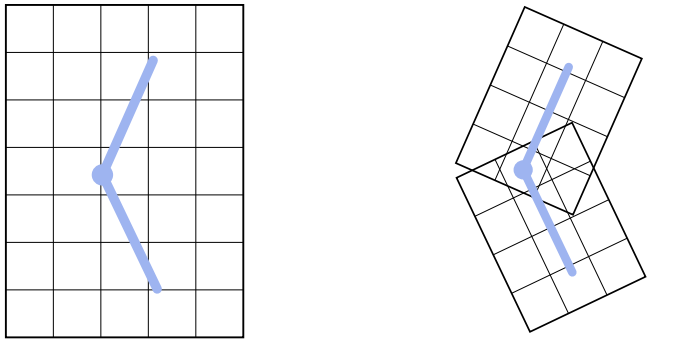
where $e(\mathbf{u}) = D(\mathbf{u}) - \hat{D}(\mathbf{u})$ is the projective signed distance error between the noisy observation and true surface point. In practice, we use a likelihood function that is simpler to maximize; we create a point cloud from the depth map, and treat each resulting 3D point as if it were generated by the closest point on the model surface, with one-dimensional additive Gaussian noise in the direction of the surface normal at that point. To create the point cloud, for each pixel $\mathbf{u}$ in an observed depth map, we back-project the value to its corresponding point in the camera frame of reference $\mathbf{x}_\mathbf{u} \in \mathbf{R}^3$ according to

$$\mathbf{x}_\mathbf{u} = D(\mathbf{u})\mathbf{K}^{-1}(u, v, 1)^\top \,, \tag{2.5}$$

where $\mathbf{K}$ defines the depth camera intrinsic calibration matrix with known focal length and principle point parameters.

Our simplified likelihood function amounts to an articulated iterative closest point (ICP) objective function, such as is solved by Dewaele, Devernay, and Horaud, (2004), Grest, Woetzel, and Koch, (2005), Ye and Yang, (2014), and others. Fitzgibbon, (2003) showed that the use of implicit surface representations in the form of precomputed signed distance functions removes the need for the explicit nearest point computation in the traditional rigid ICP algorithm, replacing it with an extremely fast look-up in the SDF. We observe that the same can be done for articulated ICP, given an articulated model signed distance function, $\mathrm{SDF}_{\mathrm{mod}}(\mathbf{x}_\mathbf{u}; \theta) : \mathbb{R}^3 \to \mathbb{R}$, which defines for all of $\mathbb{R}^3$ the distance to the surface of the model when posed according to parameter vector θ. Assuming such a function exists and has been precomputed, the distance from the back-projected point at pixel $\mathbf{u}$ to the the closest point on the model surface can be found simply by looking up the value stored in the signed distance function at that point. The likelihood of the observed depth map then becomes

$$p(D|\theta) \propto \prod_{\mathbf{u}\in\Omega} \exp\left(-\,\mathrm{SDF}_{\mathrm{mod}}(\mathbf{x}_\mathbf{u}; \theta)^2/\sigma^2\right) . \tag{2.6}$$



(A) Global SDF (B) Local SDF Ensemble

FIGURE 2.3: A visualization of two approaches to computing signed distance functions for a simple articulated model consisting of two coordinate frames linked by a rotational joint. (A) A single, global SDF is computed, giving the distance from any point within the voxel grid to the nearest point on the model surface. However, this is pose-specific and will need to be recomputed if the joint rotates. (B) Two separate, local SDFs are computed, one for each coordinate frame. Computing the distance to the closest point on the model surface now involves two look-ups, but because each SDF is represented in a local coordinate frame, model motion does not necessitate any SDF recomputation.

The maximum likelihood estimate (MLE) of θ is then found by minimizing the negative log of equation 2.6, yielding

$$\hat{\theta} = \arg\min_{\theta} \sum_{\mathbf{u}\in\Omega} \left(\mathrm{SDF}_{\mathrm{mod}}(\mathbf{x}_{\mathbf{u}};\theta)\right)^2 . \tag{2.7}$$

A single precomputed SDF suffices for rigid body tracking, but when the model is articulated the distance function is a function of the state. This means the global function $\mathrm{SDF}_{\mathrm{mod}}(\mathbf{x};\theta)$ can no longer be precomputed. Furthermore, recomputing a full global SDF every time the pose estimate changes would be computationally expensive.

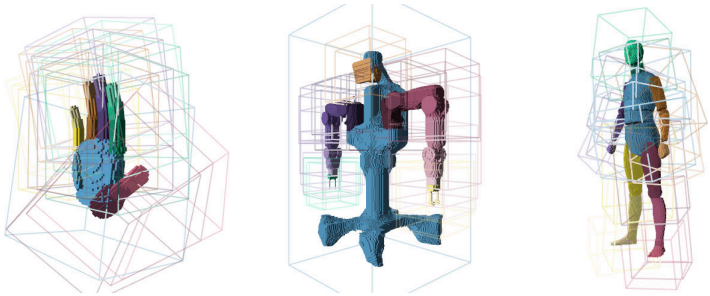
Instead, we solve this problem by approximating the global signed distance function, $\mathrm{SDF}_{\mathrm{mod}}(\mathbf{x};\theta)$, as a composition of the precomputed local signed distance functions $\mathrm{SDF}^i(\mathbf{x})$ for each frame of reference i to which geometry is rigidly attached. Figure 2.3 illustrates this principle. To do the composition, we first define the data association of a point $\mathbf{x}_{\mathbf{u}}$ relative to the camera frame of reference as

$$k^*_{\mathbf{u}}(\theta) = \arg\min_{k\in\mathcal{M}} \mathrm{abs}\Big(\mathrm{SDF}^k\left(T_{k,c}(\theta) * \mathbf{x}_{\mathbf{u}}\right)\Big) . \tag{2.8}$$

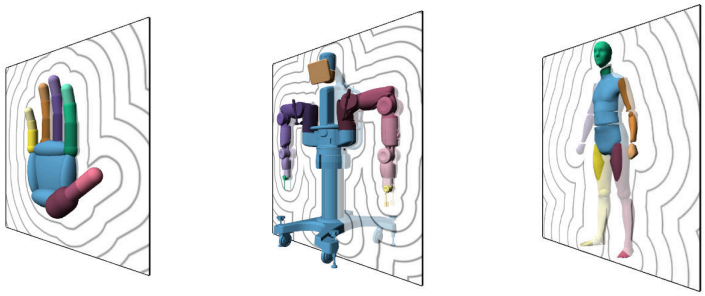
In words, we simply look up the SDF value for all rigid components of the model and associate the pixel with the geometry fragment which will lead to the minimal error with the current pose estimate. We can then define the global SDF for any point via the data association:

$$e_{\mathbf{u}}(\theta) := \mathrm{SDF}_{\mathrm{mod}}(\mathbf{x}_{\mathbf{u}};\theta) = \mathrm{SDF}^{k^*}\left(T_{k^*,c}(\theta) * \mathbf{x}_{\mathbf{u}}\right) . \tag{2.9}$$

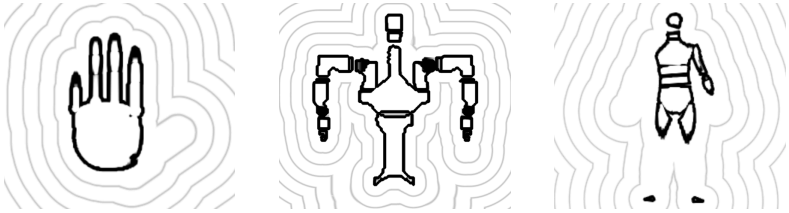
This strategy of computing the global SDF dynamically via a composition of local SDFs involves a trade-off between the need to constantly recompute the global SDF and



(A) Voxelized models of a hand, Baxter robot, and human body.



(B) Slices through the models along with isocontours of the composite SDF.



(C) Contours of the composite SDFs, computed via equation 2.9.

(D) Part associations computed via equation 2.8.

FIGURE 2.4: Representation of articulated models in DART. Each rigid component of the model is voxelized, and a signed distance function is computed in the local coordinate frame. (A) shows bounding boxes of the discrete SDFs (each color is a separate part) with interior voxels filled in. (B) shows a planar slice through the models with the isocontours of the composite global SDF superimposed on the plane. This plane is shown in an orthogonal view in (C) with zero level sets in bold (As a 3D distance transform, the isocontours of the slices are influenced by out-of-plane geometry, such as by the thumb at left). (D) uses the part colors (as seen in the second row) to indicate the rigid part used to generate the composite global SDF value at each plane location.

the need to perform multiple SDF look-ups. There are a number of reasons to favor the latter. First, the fact that the SDFs can be entirely precomputed means that the run-time performance is not impacted by the SDF resolution (other than indirectly via cache performance), which is not the case if the global SDF has to be recomputed for each new pose estimate. Second, decomposing the model into smaller segments and computing an SDF for each often allows for a tighter fit of the SDF volumes to the model, which can help save memory (c.f. Figure 2.3). The need to look up multiple SDF values is also not as bad as it might seem; in most cases the number of rigid components is relatively small, and as the model grows a quick bounding box check can quickly rule out most components. Finally, the number of SDF look-ups that we incur is independent of the size of the model and the complexity of the surface geometry, which is an advantage over point-based algorithms such as ICP. Complex surfaces would require many points to represent faithfully, which increases the cost of computing the closest point correspondences.

In Figure 2.4 we illustrate a selection of global SDF compositions for a variety of models, each in a characteristic pose. For each model, a 2D slice through the SDF is visualized along with the associated closest part. The left column of Figure 2.5 illustrates the data association, SDF error, and gradients induced by an observation in the composite model SDF.

2.2.3 Optimization

Inserting the per-pixel error defined in equation 2.9 into the MLE of equation 2.7, we arrive at the approximated articulated tracking energy,

$$\hat{\theta} = \arg\min_{\theta} \sum_{\mathbf{u}\in\Omega} e_{\mathbf{u}}(\theta)^2 \,. \tag{2.10}$$

As this is a standard sum of squared errors, we are able to use Gauss-Newton optimization to minimize the negative log likelihood, which iteratively finds the step $\Delta\theta$ in parameter space which, according to a local linearization around the current estimate, will lead to a minimum in the objective function. That is, we want to find:

$$\hat{\Delta\theta} = \arg\min_{\Delta\theta} \sum_{\mathbf{u}\in\Omega} e_{\mathbf{u}}(\theta \oplus \Delta\theta)^2 \,. \tag{2.11}$$

where $\oplus$ is the operator that composes the parameter update vector $\Delta\theta$ by mapping the local parameterization of the update into the ambient parameter space. This step is given by:

$$\hat{\Delta\theta} = -\left(\sum_{\mathbf{u}\in\Omega} J_{\mathbf{u}}(\theta)^\top J_{\mathbf{u}}(\theta)\right)^{-1} \left(\sum_{\mathbf{u}\in\Omega} J_{\mathbf{u}}(\theta)^\top e_{\mathbf{u}}(\theta)\right) \,, \tag{2.12}$$

where $J_{\mathbf{u}}(\theta) = \frac{\partial}{\partial\Delta\theta} e_{\mathbf{u}}(\theta \oplus \Delta\theta)\Big|_{\Delta\theta=0}$. The state is updated using:

$$\theta \leftarrow \theta \oplus \Delta\theta \,, \tag{2.13}$$

concluding the iteration. This process of linearizing the objective function, computing a parameter update, and composing the update is iterated until some convergence criteria have been met.

We will now discuss the computation of first derivatives used to solve equation 2.12. In each iteration, we compute the data association $k^*_{\mathbf{u}}(\theta)$ and error $e_{\mathbf{u}}(\theta)$ as given by equations 2.8 and 2.9. The computation of $J_{\mathbf{u}}(\theta)$ breaks down nicely via the chain rule:

$$\begin{aligned} \frac{\partial}{\partial \Delta\theta} e_{\mathbf{u}}(\theta \oplus \Delta\theta)|_{\Delta\theta=0} = \\ \frac{\partial}{\partial \Delta\theta} \text{SDF}^{k^*_{\mathbf{u}}}(T_{k^*_{\mathbf{u}},c}(\theta \oplus \Delta\theta) * \mathbf{x}_{\mathbf{u}}) = \\ \nabla \text{SDF}^{k^*_{\mathbf{u}}}(T_{k^*_{\mathbf{u}},c}(\theta) * \mathbf{x}_{\mathbf{u}}) \frac{\partial}{\partial \Delta\theta} \left(T_{k^*_{\mathbf{u}},c}(\theta \oplus \Delta\theta) * \mathbf{x}_{\mathbf{u}} \right) , \end{aligned} \tag{2.14}$$

where we simply need to compute a $3 \times N$ matrix which describes the motion of point $\mathbf{x}_{\mathbf{u}}$ in $\mathbb{R}^3$ caused by incremental pose update $\Delta\theta$, assuming the point is fixed in the local frame of reference $k^*_{\mathbf{u}}$. We then compute the dot product with the local SDF gradient at the current point location.

Given our model parameterization, the transform from the global frame c to $k^*_{\mathbf{u}}$ follows some chain of the form $T_{k^*_{\mathbf{u}},c} = T_{k^*_{\mathbf{u}},parent(k^*_{\mathbf{u}})} ... T_{0,c}$. By convention, the first 6 parameters (denoted $\theta_{1:6}$) describe the 6 degrees of freedom of the global transformation $T_{0,c}$ from the camera to the root frame of reference. Updates to this transformation are parameterized locally using the Lie algebra $\mathfrak{se}(3)$, which has been shown to be an effective representation for gradient-based visual tracking techniques (Drummond and Cipolla, 1999; Drummond and Cipolla, 2001).

As is typical, we would like to compute gradients of the global pose around the identity transform. To do this, we'll rely on the the reparameterization[2]

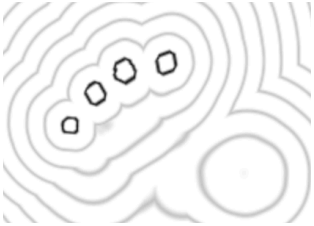
$$T_{0,c}(\theta \oplus \Delta\theta) = T_{0,0'}(\Delta\theta) * T_{0',c}(\theta) . \tag{2.15}$$

Here we have instantiated an intermediate root frame called $0'$ which initially has the same pose as the root frame, i.e. $T_{0,0'}(\mathbf{0}) = I$. This parameterization results in a matrix

[2]We could also have reparameterized using a right-multiplication of the update matrix, which would be mathematically equivalent given that initially $\Delta\theta = \mathbf{0}$. However, the gradients for this parameterization are different, and the gradients for left-multiplication are better conditioned for optimization as they describe transformations in the object frame of reference rather than the camera frame. To understand why this is desirable, imagine an object observed from a distance of a meter. In this pose, the direction of motion of a point on the object caused by an infinitesimal rotation about the camera's y axis and that caused by an infinitesimal translation about the camera's x axis will by nearly identical (assuming z is the optical axis). The effect is exacerbated as the object becomes more distant from the camera, and in the limit as the distance approaches infinity the Hessian approximation (i.e. $J^\top J$) approaches a matrix with a null space of rank 2 (the same issue occurs with the x and y axes swapped). In practice, this can also cause problems at a moderate distance, especially if the object rotates about an axis through its own center of mass parallel to the camera's y axis; in this case, the optimization must describe this rotation according to right-multiplication using a rotation about the y axis and a relatively large translation to re-center the object. With left multiplication, this motion can be described using rotation only. This is a particular important consideration for objects such as human hands which are highly dynamic and can rotate rapidly.

(A) Observed point cloud

(B) Predicted point cloud

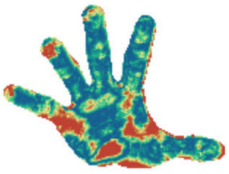


(C) Model SDF planar slice

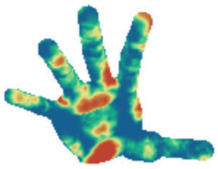
(D) Observation SDF planar slice

(E) Observed association

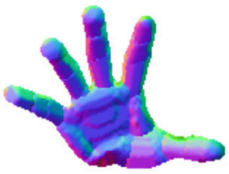
(F) Prediction association



(G) Observed point errors



(H) Predicted point errors



(I) Observed point normals

(J) Predicted point normals

FIGURE 2.5: (A) and (B) show the observed and predicted point clouds for a given depth frame, respectively, along with the model skeleton. (C) shows a planar slice of the model SDF, and (D) shows a slice of the observation SDF on the same plane (black indicates the zero level set). Each observed point in (A) is looked up in the model SDF (C) to determine the rigid part data association (E), error (G), and SDF gradients (I) which are used to compute derivatives of the energy function. Likewise, each predicted point in (B) with known rigid part data association (F) is looked up in the observation SDF (D) to determine errors (H), and gradients (J).

of the following form:

$$\frac{\partial}{\partial \Delta\theta}\left(T_{k^*_{\mathbf{u}},c}(\theta \oplus \Delta\theta) * \mathbf{x} = \right) \\ \left(R_{k^*,0}(\theta) * \frac{\partial}{\partial \Delta\theta} T_{0,0'}(\Delta\theta) * T_{0',c}(\theta) * \mathbf{x} \;\middle|\; \frac{\partial \mathbf{x}}{\partial \Delta\theta_7} \quad \cdots \quad \frac{\partial \mathbf{x}}{\partial \Delta\theta_N} \right) , \tag{2.16}$$

where $R_{k^*_{\mathbf{u}},0}(\theta)$ is the rotational component of the transform from root frame 0 to $k^*_{\mathbf{u}}$. The first 3×6 block can be computed, following section 10.3.5 of Blanco, (2013), using

$$\frac{\partial}{\partial \Delta\theta} T_{0,0'}(\Delta\theta) * T_{0',c}(\theta) * \mathbf{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & x^0_3 & -x^0_2 \\ 0 & 1 & 0 & -x^0_3 & 0 & x^0_1 \\ 0 & 0 & 1 & x^0_2 & -x^0_1 & 0 \end{pmatrix} , \tag{2.17}$$

where $\mathbf{x}^0 = T_{0,c} * \mathbf{x}$ and $\mathbf{x} = [x_1 \; x_2 \; x_3]^\top$. To compute $\frac{\partial}{\partial \Delta\theta_i} T_{k^*_{\mathbf{u}},c}\mathbf{x}_{\mathbf{u}}$ for the columns corresponding to joint parameters we apply the product rule and, assuming each element of the pose vector can affect at most one transform, we get at most one nonzero term of the form:

$$\frac{\partial}{\partial \Delta\theta_i} T_{k^*_{\mathbf{u}},c}(\theta \oplus \Delta\theta) * \mathbf{x}_{\mathbf{u}} = \\ R_{k^*_{\mathbf{u}},parent(j)}(\theta) \left(\frac{\partial}{\partial \Delta\theta_i} T_{parent(j),j}(\theta \oplus \Delta\theta) \right) T_{j,c}(\theta) * \mathbf{x}_{\mathbf{u}} , \tag{2.18}$$

where j is the frame of reference whose transformation (relative to its parent) depends on θ_i. If θ_i describes a rotational joint with some axis $\mathbf{z}$ in the frame of reference $parent(j)$, then this becomes:

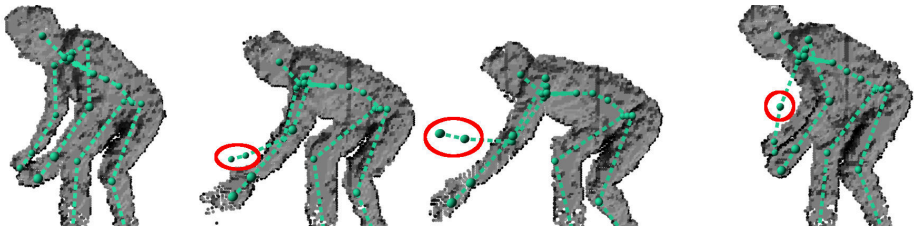
$$\frac{\partial}{\partial \Delta\theta_i} T_{k^*_{\mathbf{u}},c}(\theta \oplus \Delta\theta) * \mathbf{x}_{\mathbf{u}} = R_{k^*_{\mathbf{u}},parent(j)}(\theta) \left(\mathbf{z} \times T_{parent(j),c}(\theta) * \mathbf{x}_{\mathbf{u}} \right) . \tag{2.19}$$

If θ_i describes a prismatic joint along some axis $\mathbf{z}$, also in the frame of reference of $parent(j)$, then we simply have:

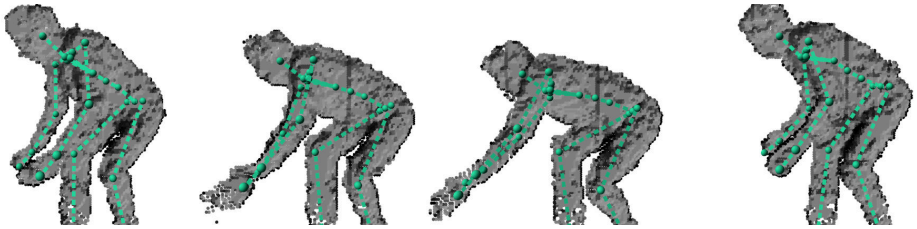
$$\frac{\partial}{\partial \Delta\theta_i} T_{k^*_{\mathbf{u}},c}(\theta) * \mathbf{x}_{\mathbf{u}} = R_{k^*_{\mathbf{u}},parent(j)}(\theta)\mathbf{z} . \tag{2.20}$$

The above two situations only apply when the pose of frame $k^*_{\mathbf{u}}$ depends on parameter θ_i, which is to say that θ_i affects some transform between frame $k^*_{\mathbf{u}}$ and the root; if, on the other hand, θ_i appears below $k^*_{\mathbf{u}}$ or in another branch of the kinematic tree, then column i of $J_{\mathbf{u}}(\theta)$ will be the zero vector.

Once we have computed the incremental update we must perform the composition of equation 2.13. As stated previously, $\theta_{1:6}$ describes the global 6 DoF transformation between the camera and the model via the exponential map from $\mathfrak{se}(3)$ to $\mathbb{SE}(3)$ (c.f. Blanco, (2013), section 9.4.2). Given that we've solved for a relative transformation about our current estimate, we update this parameter block by composing the transformation



(A) Without Free Space Information



(B) With Free Space Information

FIGURE 2.6: Stills from human body tracking, demonstrating the benefits of using free space information. (A) DART without free space optimization loses track of the occluded arm, as indicated by the skeleton protruding into free space (grossly mispredicted joints are circled in red). (B) Using free space information allows DART to maintain a reasonable estimate through occlusion, which is still within the basin of convergence when the arm reappears in the depth map.

from the camera to the root frame of reference according to:

$$T_{0,c} \leftarrow e^{\Delta\theta} T_{0,c} \,, \quad \epsilon = \begin{bmatrix} 0 & -\Delta\theta_6 & \Delta\theta_5 & \Delta\theta_1 \\ \Delta\theta_6 & 0 & -\Delta\theta_4 & \Delta\theta_2 \\ -\Delta\theta_5 & \Delta\theta_4 & 0 & \Delta\theta_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} . \tag{2.21}$$

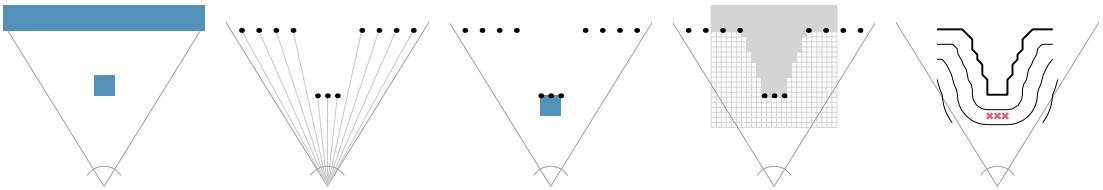
The composition of the remainder of the parameter vector can be done with simple vector addition:

$$\theta_{7:N} \leftarrow \theta_{7:N} + \Delta\theta_{7:N} \,. \tag{2.22}$$

Crucially, our dense, fully-articulated model tracking algorithm described here trivially parallelizes onto modern GPU hardware. We'll elaborate on GPU implementation details in section 2.3.

2.2.4 Free Space Constraints and the Observation SDF

Traditional ICP-based approaches neglect important information available in depth observations; namely, each pixel in a depth map provides information not only about where surfaces exist in the world, but also where there are none. Any observation of a nonzero depth indicates that there is nothing between that observed point and the camera, up to some noise threshold and barring sensor error. The utility of this 'negative' information was noted by Ganapathi et al., (2012), who used constraint projection to avoid gross free space violations in their model predictions.



(A) (B) (C) (D) (E)

FIGURE 2.7: A 2D illustration of how the observation SDF is constructed and why it is useful. (A) A simple scene in which a camera (in gray) observes a box before a flat background. (B) The back-projected point cloud from the observation such a camera would make in this scene, along with the rays along which the points were observed. Imagine our current pose estimate is as shown in (C), for whatever reason (e.g. swift motion of the box away from the camera and towards the wall) – under the error function in (2.10), this would be a valid solution, motivating the observation SDF. (D) shows a finite grid within which the observation SDF for this observation will be computed. Grid cells which project behind an observed point are shown in grey, and will be initialized to zero. After a distance transform operation, this initialization will lead to the isocontours shown in (E). With the additional free space term in the objective function, as given in equation 2.25, the estimate in (C) is no longer satisfactory, as shown by superimposing the points we would predict to see given this pose estimate on the observation SDF isocontours. The same applies to the 3D setting, with the observation SDF penalizing model parts protruding in front of or to the side of the observed point cloud.

To incorporate negative information, we compute a signed distance function representation of each observation, $\text{SDF}_{\text{obs}}(\mathbf{x}; D)$, and add a simple complementary term to the model SDF energy described in the previous section, which we call the observation SDF term. We are then able to take negative information into account directly in the optimization, rather than relying on constraint projection. The result is a nicely symmetric objective function, where observed points incur an error in the predicted model SDF, and predicted points incur an error in the observation SDF. A model surface which is predicted to lie in front of or to the side of the observation will induce an error in this complementary term which can be reduced by moving the surface out of known free space. In addition to the elegant symmetry, we find the addition results in superior tracking robustness, as can be seen in Figure 2.6.

Specifically, we first instantiate a volume of fixed size such that it entirely encloses our model in the current frame. We compute the pixel onto which each voxel projects and set its value to zero if it has depth greater than the observation, and to a large positive value otherwise. We then compute the 3D distance transform to obtain $\text{SDF}_{\text{obs}}(\mathbf{x}; D)$. Figure 2.7 gives a 2D visual overview of the construction and use of the observation SDF, and an illustration of the effect on a real observation is shown in the right column of Figure 2.5.

For each iteration of the optimization we generate a predicted point cloud by rendering the model geometry according to the current pose estimate. This is done with a standard OpenGL rendering pipeline and a shader which also produces a map of data association labels $\tilde{K}$ identifying the rigid body from which each point was generated.

Then, we can augment our probability distribution in equation 2.3:

$$p(D|\theta) = \prod_{\mathbf{u}\in\Omega} p(D(\mathbf{u})|\theta)p(\tilde{\mathbf{x}}_{\mathbf{u}}(\theta)|D) \,, \tag{2.23}$$

where $\tilde{\mathbf{x}}_{\mathbf{u}}(\theta)$ is the rendered vertex at pixel location $\mathbf{u}$. Then the likelihood function for the predicted vertex map is defined by

$$p(\tilde{\mathbf{x}}_{\mathbf{u}}(\theta)|D) \propto \exp(-\mathrm{SDF}_{\mathrm{obs}}\left(\tilde{\mathbf{x}}_{\mathbf{u}}(\theta); D\right)^2) \,. \tag{2.24}$$

Finally, we arrive at a new energy minimization of the form:

$$\hat{\theta} = \arg\min_{\theta} \sum_{\mathbf{u}\in\Omega} \mathrm{SDF}_{\mathrm{mod}}\left(\mathbf{x}_{\mathbf{u}}; \theta\right)^2 + \lambda \sum_{\mathbf{u}\in\Omega} \mathrm{SDF}_{\mathrm{obs}}\left(\tilde{\mathbf{x}}_{\mathbf{u}}(\theta); D\right)^2 \tag{2.25}$$

which expresses both the positive and negative information available in the observation, and balances the two terms. The balancing factor λ has a probabilistic interpretation, in that it represents the ratio of the variance between the two measurement models, and given that the variance on each term is in units of distance in $\mathbb{R}^3$, it has a geometric interpretation as well. Note that the cost of computing $\mathrm{SDF}_{\mathrm{obs}}\left(x; D\right)$ is independent of the number of degrees of freedom and must be done only once per frame. The error induced in the observation by the current model rendering is computed by direct trilinear interpolation of the SDF, as with the model SDF. For the derivatives, we note that for each predicted point $\tilde{\mathbf{x}}_{\mathbf{u}}$ there is some (fixed) point $\tilde{\mathbf{x}}'_{\mathbf{u}}$ in frame of reference $\tilde{k}$ such that

$$\tilde{\mathbf{x}}_{\mathbf{u}}(\theta) = T_{c,\tilde{k}}(\theta) * \tilde{\mathbf{x}}'_{\mathbf{u}} \,. \tag{2.26}$$

In this case, the correspondence $\tilde{k}$ is known. We thus compute $\tilde{\mathbf{x}}'_{\mathbf{u}}$ for each point in the predicted point cloud and compute $\tilde{J}_{\mathbf{u}}(\theta) = \frac{\partial}{\partial\Delta\theta}\mathrm{SDF}_{\mathrm{obs}}\left(T_{c,\tilde{k}}(\theta \oplus \Delta\theta)\tilde{\mathbf{x}}'_{\mathbf{u}}; D\right)$, which is done exactly as in equations (2.14) and (2.16), excepting that the transform is inverted.

The ease with which this new error term can be incorporated into the existing objective function highlights the modularity that is common to many generative model-based techniques as discussed in Chapter 1. We'll explore further additions to the objective function in Chapter 3.

2.3 GPU Implementation

We will now discuss some of the key design decisions and engineering choices that allow us to track highly articulated models in real-time with dense data. The key is our use of the CUDA parallel computing platform[3], which allows us to define a parallel task (called a CUDA kernel) and then launch thousands of threads to execute the kernel independently in parallel. The most important difference between GPU and CPU programming is that many parallel algorithms which are computation-bound on the CPU become memory-bound when implemented on the GPU, as access to global GPU memory and data transfers between the CPU and GPU are extremely slow compared to the speed at which the independent threads can compute. Our implementation of the DART tracking algorithm was designed with this architecture in mind. We will now

[3]https://developer.nvidia.com/cuda-zone

FIGURE 2.8: The flow of the DART tracking algorithm illustrated with a real instance of pose estimation for the Baxter robot. Starting with the initial estimate and depth map as input (lower left), we first compute the corresponding point cloud and observation SDF. We then iteratively refine the pose estimate, as shown on the right. The observation to model data association, the corresponding error, and the resulting normal equations can be computed in parallel with the model to observation data association, error, and normal equations. The normal equations are then combined to update the pose estimate. This process is repeated until convergence or until the next frame arrives. The final pose estimate along with its uncertainty is used as the initial pose estimate for the next frame, possibly after updating it through a dynamics model.

describe the CUDA kernels that are core to the speed and functionality of the DART tracker.

The pipeline from a depth map to an articulated pose is as follows: first the depth map is processed to generate a 2D-indexed vertex map or point cloud, and the observation SDF. Next, the association (if any exists) of all pixels with a frame in the kinematic chain is determined. Then, the contribution of each data-associated point to the normal equations is calculated. Finally, these contributions are summed and sent back to the CPU, where the normal equations are solved to determine the pose estimate update. The flow of the tracking algorithm is illustrated on a real tracking example in Figure 2.8.

2.3.1 Depth Map Processing

When a depth map arrives from the sensor, it is sent immediately to the GPU, where all further processing takes place; the next transfer between GPU and CPU is not until the normal equations are sent back.

The first step is to generate a point cloud from the depth map, described mathematically in equation 2.5. A CUDA kernel produces a vertex map stored as a 2D-indexed grid of floating point vectors of length four, where the first three values represent the 3D point and the final value represents a binary indication of point validity. Each thread is responsible for producing one point in the point cloud from one pixel in the depth map. We generally assume a pinhole camera model such that the calibration is entirely specified by the focal length and principal point of the sensor, although more complicated sensor calibration could be implemented as well.

For each new depth map we must also compute a new observation SDF as described in section 2.2.4. This is a two step process in which the SDF is first initialized to a binary function indicating whether or not each voxel lies in free space, and then turned into a distance function via a 3D distance transform as described by Felzenszwalb and Huttenlocher, (2012). This is implemented with two CUDA kernels, the first performing the initialization with one thread responsible for each voxel, and the second a standard 3D distance transform. Note that, while $\mathrm{SDF}_{\mathrm{obs}}(\mathbf{x})$ is in theory defined as a continuous function of $\mathbf{x} \in \mathbb{R}^3$, it is stored as a discrete function $\mathrm{SDF}_{\mathrm{obs}}(\mathbf{v})$ of $\mathbf{v} \in \mathbb{Z}^3$, and the continuous function is approximated using trilinear interpolation. After the distance transform, the grid will store in each voxel the distance to the nearest voxel that was initialized to zero by the first kernel.

2.3.2 Data Association

Once the point cloud and observation SDF have been computed, the iterative solver begins optimizing the pose. The first step of this optimization is to compute the pixel-wise data association and error as defined by equations 2.8 and 2.9. This is done with a CUDA kernel with each thread responsible for the data association of one pixel, as described in Algorithm 1. Unlike dense camera tracking scenarios in which almost the entire image can be used for tracking, we are here interested in tracking foreground models that often project onto a relatively small portion of the input image. We use a model distance threshold τ_d to reject data association of background pixels, and as a result only a fraction of pixels in a typical depth map will be associated with a rigid body. In order to make the next kernel more efficient, we would like the data association information to be computed as a dense, linearly indexed array rather than a sparse,

Algorithm 1: Data association kernel

input : pixel index $\mathbf{u}$, vertex map $V(\mathbf{u})$, model $\mathcal{M}$, camera-to-rigid-body transforms $T_{i,c}$ for $i \in \mathcal{M}$, signed distance function $SDF^i(\mathbf{x})$ for $i \in \mathcal{M}$, distance threshold τ_d, index l to last item in arrays

output: data association array $k(i)$, distance array $d(i)$, pixel index array $u(i)$

```
x_c := V(u);
if x_c.w = 1 then
    k_u := −1 ;
    d_u := τ_d ;
    for i ∈ M do
        x_i := T_{i,c} x_c;
        if x_i in bounds of SDF^i then
            d_i := trilinear interpolation of SDF^i(x_i);
            if d_i < d_u then
                d_u ← d_i;
                k_u ← i;
            end
        end
    end
    if k_u >= 0 then
        i_u := atomicAdd(l, 1);
        k(i_u) ← k_u;
        d(i_u) ← d_u;
        u(i_u) ← u;
    end
end
```

2D-indexed grid. The traditional way to do this would be to compute the sparse grid first, then use a parallel stream compression algorithm on the GPU to remove unassociated pixels. However, we found that with modern GPU hardware and software, it was actually faster to compute the dense array directly by maintaining an index for the last element in the array, which each thread increments atomically if the pixel for which it is responsible gets associated to a rigid body. The atomic add operation returns the index to which the thread writes its association data, namely the index of the point which has been associated, the rigid body to which it has been associated, and the distance between the point and the associated rigid body surface.

Rather than computing the recursive transforms $T_{k,c}(\theta)$ used in equation 2.8 separately for each pixel, we first compute the transform from the camera frame of reference to the frame of reference of each rigid body k on the CPU, then copy the values to the GPU, allowing them to be re-used by all threads of the data association kernel.

Algorithm 2: normal equations kernel

input : associated point index i, data association array $k(i)$, distance array $d(i)$, pixel index array $u(i)$, vertex map $V(\mathbf{u})$, model $\mathcal{M}$, camera-to-root transform $T_{0,c}$, root-to-rigid-body transforms $T_{i,0}$ for $i \in \mathcal{M}$, signed distance function $\text{SDF}^i(\mathbf{x})$ for $i \in \mathcal{M}$

output: Hessian approximation $J^\top J$, gradient direction $J^\top e$

$k := k(i)$;
$d := d(i)$;
$\mathbf{u} := u(i)$;
$\mathbf{x_c} := V(\mathbf{u})$;
$\mathbf{x_0} := T_{0,c}\mathbf{x_c}$;
$\mathbf{x_k} := T_{k,0}\mathbf{x_0}$;
$\mathbf{g} :=$ trilinearly interpolated central differences of $\text{SDF}^k(\mathbf{x_k})$;
$J(1) \leftarrow \mathbf{g} \cdot [1, 0, 0]^\top$;
$J(2) \leftarrow \mathbf{g} \cdot [0, 1, 0]^\top$;
$J(3) \leftarrow \mathbf{g} \cdot [0, 0, 1]^\top$;
$J(4) \leftarrow \mathbf{g} \cdot [0, -\mathbf{x_0}.z, \mathbf{x_0}.y]^\top$;
$J(5) \leftarrow \mathbf{g} \cdot [\mathbf{x_0}.z, 0, -\mathbf{x_0}.x]^\top$;
$J(6) \leftarrow \mathbf{g} \cdot [-\mathbf{x_0}.y, \mathbf{x_0}.x, 0]^\top$;
for *joint* $j \in \mathcal{M}$ **do**
  if *j is rotational* **then**
    $\mathbf{x_j} := T_{frame_j,0}\mathbf{x_0}$;
    $J(6+j) \leftarrow \mathbf{g} \cdot (\mathbf{x_j} \times axis_j)$;
  else if *j is prismatic* **then**
    $J(6+j) \leftarrow \mathbf{g} \cdot axis_j$;
  end
end
for $c = 0$ **to** *number of degrees of freedom* **do**
  atomicAdd$(J^\top e_r, J(r)d)$;
  for $r = 0$ **to** c **do**
    atomicAdd$(J^\top J_{rc}, J(r)J(c))$;
  end
end

2.3.3 Normal Equations

The next step is to compute the normal equations used to solve for the parameter update, as given in equation 2.12. The normal equations consist of a Hessian approximation, $J^\top J$, and a gradient step direction $J^\top e$. Each data associated pixel will contribute to both of these values. Traditionally, we would compute the individual contribution of each pixel, store them in GPU memory, then do a tree reduction to compute the sum. However, we again found it faster on modern hardware to simply compute the sum directly using atomic addition operations, as shown in the kernel pseudocode in Algorithm 2.

To compute the contribution of each pixel, we first read in the data association and error as produced by the data association kernel. Then we compute the first derivative of the error according to the model structure and current pose estimate, and finally compute the values of $J^\top J$ and $J^\top e$ and add them atomically to the global sum. Due to the fact that the Hessian approximation is necessarily symmetric, we compute only the upper triangle of the matrix.

2.3.4 Negative Information

For the negative information term of section 2.2.4, the data association step is unnecessary as we already know the rigid body from which each predicted point was generated. We are then able to compute $J^\top J_{\text{obs}}$ and $J^\top e_{\text{obs}}$ using a kernel nearly identical to that described in Algorithm 2. We can then simply blend the Hessian approximations and gradient steps via:

$$\begin{aligned} J^\top J &= J^\top J_{\text{mod}} + \lambda J^\top J_{\text{obs}} \\ J^\top e &= J^\top e_{\text{mod}} + \lambda J^\top e_{\text{obs}} \,, \end{aligned} \tag{2.27}$$

which is done on the CPU.

2.3.5 Pose Update

Solving the normal equations generally involves a relatively small matrix inversion ($N \times N$), and is therefore handled efficiently enough on the CPU. We then compose the resulting step $\Delta\theta$ onto our pose estimate, recompute the camera-to-frame transforms, and iterate as needed.

2.4 Experimental Evaluation

As the key focus of DART is the ability to generalize across a wide variety of models, we evaluate its performance in multiple domains. We give quantitative results for markerless tracking from depth datasets in the human body and human hand domains, and demonstrate a robotics application that relies on accurate tracking of a robot manipulator. There are some variations in parameter values to account for differences in model size and sensor types, but the code used to track all models is the same. Results are given for both the symmetric and asymmetric formulations of DART, which refer to the presence or absence of the free space constraints described section 2.2.4, respectively. All results were computed using depth maps at a 320x240 pixel resolution at a frame rate at or above 30 FPS on an Nvidia GeForce GTX 680 with minimal CPU requirements. Model SDFs are generally computed at a resolution of anywhere between 2 mm and 8 mm per side, depending on the model size and the highest-frequency features needed for tracking. As noted earlier, this is a precomputation step and only affects performance insofar as it affects the caching behavior of SDF look-ups. The individual SDFs are generally computed out to anywhere between 7 cm and 15 cm beyond the strict geometry bounds. There are diminishing returns beyond these distances as data association typically becomes less meaningful with increased distance from the surface.

Ganapathi et al.	DART asym.	DART symm.	Ye and Yang
0.841	0.873	0.894	0.921

TABLE 2.1: Comparison with state-of-the-art human body tracking approaches on the EVAL dataset of Ganapathi et al., (2012). Results given are the percentage of joints which are predicted within 10 cm of the given ground truth positions.

2.4.1 Human Body Tracking

First, we demonstrate the performance of DART on the EVAL dataset presented by Ganapathi et al., (2012), and compare our results with the results from that work as well as the results of Ye and Yang, (2014). The human body models used are based on the meshes of Helten et al., (2013), and had 48 degrees of freedom. The dataset consists of a sequence of point clouds with ground truth joint markers derived from a motion capture system. The error metric used is the percentage of joints predicted within 10 cm of the given ground truth across all 24 sequences in the dataset. We'd like to note that a close analysis of this data set revealed that there is non-negligible noise in the ground truth marker positions themselves, thereby making a perfectly accurate evaluation impossible.

The results presented in Table 2.1 show that DART is competitive with state-of-the-art human body tracking approaches. We are able to improve on the results of Ganapathi et al., (2012) even without using negative information, most likely because our precomputation of the model SDF allows us to use complex mesh models rather than being restricted to geometric primitive approximations. Further gains when negative information is considered are illustrated in Figure 2.6, which shows a short sequence tracked with both asymmetric and symmetric DART models.

The gap between DART and the work of Ye and Yang, (2014) on this dataset is likely explained by their further model improvements achieved by automated shape estimation, as well as their use of a smooth mesh deformation model as opposed to the segmentation of meshes into rigid parts required by DART. It is unclear how well their approach would work on other models such as human hands, as results in Table 2.2 and Figure 5.3 seem to indicate that the use of negative information has much more impact when tracking a small, agile model such as a human hand. It's also worth noting that, while our piece-wise rigidity assumption can be limiting for models that also exhibit some deformation, other objects such as robots are often truly piece-wise rigid. Furthermore, in follow-up work by Walsman et al., (2017), the DART model was extended to include surface deformations.

2.4.2 Human Hand Tracking

We also demonstrate results on the dataset of Qian et al., (2014), comparing our work against their ICP-PSO algorithm as well as the approach of Oikonomidis, Kyriazis, and Argyros, (2011). The human hand model used for DART tracking is a collection of manually-defined ellipsoids, cylinders, and spheres, and has 26 degrees of freedom. The results are shown in Table 2.2. We compare only against the reported results that do not make use of a hand-designed finger detector, as this is hand-specific. Again, DART is competitive with state-of-the-art approaches. Furthermore, this dataset clearly demonstrates the benefit of free space information incorporated via the observation SDF

Subject	1	2	3	4	5	6
FORTH	35.4	19.8	27.3	26.3	16.6	46.2
ICP-PSO	9.3	24.1	14.4	13.4	11.0	20.0
DART asym.	32.0	34.4	47.4	21.3	19.1	35.6
DART symm.	14.1	12.0	24.7	14.4	12.6	26.8

TABLE 2.2: Comparison with state-of-the-art hand tracking approaches on the dataset of Qian et al., (2014). Results are the average distance between predicted and ground truth markers for the wrist and five fingers, in millimeters. FORTH is the approach from Oikonomidis, Kyriazis, and Argyros, (2011). Results for both FORTH and ICP-PSO are as reported and implemented by Qian et al., (2014).

in the symmetric version of DART. Here, the use of such information is important for accurately tracking a model that moves as quickly and with as much self-occlusion as the human hand.

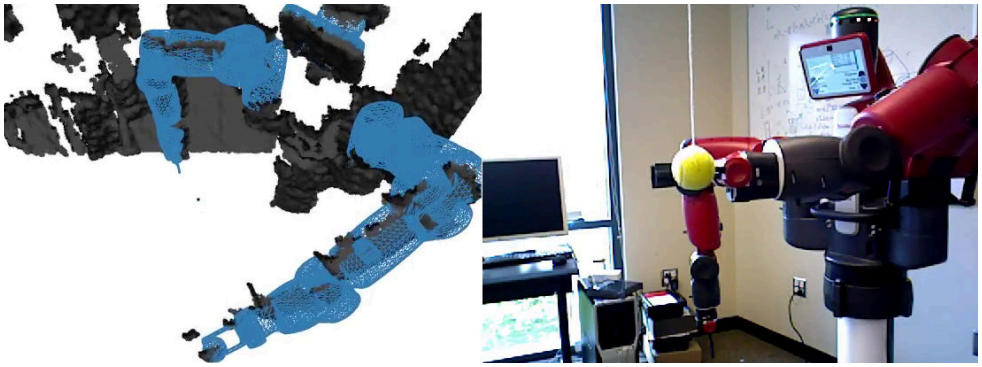
The gap between ICP-PSO and DART might be explained by the fact that the particle-based nature of PSO makes it more able to escape the sort of local minima that DART can fall into. Furthermore, the dataset was manually labeled using the model applied in ICP-PSO, which may have introduced a bias in the given marker positions. It is also unclear how well the ICP-PSO approach would generalize to human body tracking, which would require many more than 48 spheres, likely degrade performance to the point that real-time processing is not feasible. It is also highly unlikely that a sphere model could be used to track a robot with the level of accuracy achievable with our mesh model, as illustrated in the next experiment.

2.4.3 Visual Servoing

The datasets currently available for markerless tracking from a single RGB-D camera do not give a full picture of the level of accuracy achievable with a dense data term and a model of essentially arbitrary precision. In order to demonstrate its applicability to a practical robotics problem, we implemented a very simple closed-loop controller for visual servoing with the Rethink Robotics Baxter robot, for which we do have an extremely accurate mesh model. We hung a tennis ball at 10 different locations within the reach of Baxter's arms, and the task was simply to grab the ball. First, we estimated the location of the ball relative to Baxter by manually initializing the position of a Baxter and sphere model and refining the estimates with DART. We then used inverse kinematics to find joint angles required to grab a ball at the resulting relative offset and moved the arm accordingly. As a result of some combination of errors in the joint angles reported by Baxter (although the arm had just been calibrated) and depth skew in the reported depth values, the manipulator frequently went to an incorrect location and only succeeded in grasping the ball in 3 out of the 10 positions.

Next, we used DART to provide visual feedback to guide the manipulator towards the target. At fixed time intervals, the offset between the end effector position and the observed ball tracked by DART was used to refine the estimate of the ball position in Baxter's frame of reference and to update the inverse kinematics solution. When using real-time visual feedback, the ball was successfully grasped in all 10 locations. Figure 2.9 shows stills of an attempted grasp with and without visual feedback.

(A) With visual feedback



(B) Without visual feedback

FIGURE 2.9: Visual servoing results. (A) The arm position is continuously estimated, allowing for a successful grasp. (B) The position of the ball is estimated initially, but no visual feedback is used to re-estimate the manipulator's pose during motion. Though Baxter's end effector is reported to be at the initially-estimated ball position (shown in the blue overlay), it is in fact significantly off, resulting in a grasping failure.

Discussion

This chapter introduced DART, a general generative model-based framework for tracking articulated models with formally defined kinematic and geometric structure. DART represents objects via a symmetric version of signed distance functions, extended to articulated objects. The framework uses a local gradient-based optimization to find the pose of the model which best explains the data points observed in a depth camera frame along with a prior based on previous data. Because of its representation, the energy function underlying DART is fully parallelizable, and is optimized on a GPU, which allows us to use dense data to achieve accurate tracking results in real-time.

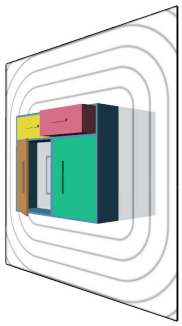


FIGURE 2.10: A cabinet model in the DART model format.

We have demonstrated successful real-time tracking in scenarios with as many as 48 degrees of freedom. Each specific application of DART only requires the specification of

a model; there are no underlying algorithmic changes required for the different objects, making it a highly portable framework. Though DART does not provide estimates that are demonstrably better than all existing methods in their specialized domains, the work is novel in its demonstration of competitive results across multiple domains. It should be noted also that DART is even more broadly applicable than what we have shown in this chapter — in Figure 2.10 we show a DART model of a cabinet with articulated doors and drawers which could be used in conjunction with other models to track the state of an entire kitchen environment.

We believe that the robotics community can greatly benefit from a general tool such as DART, enabling researchers to take advantage of accurate fine-grained information about the state of articulated objects, and at real-time update rates. As a general base to articulated model tracking, there are a multitude of ways in which DART could be extended for any particular tracking scenario. In the next chapter, we'll explore one such application.

Additional Resources

- The DART source code can be found here: https://github.com/tschmidt23/dart.
- A video showing DART in action can be found here: https://youtu.be/r40g0zvinb8.

Chapter 3

Incorporating Physical Constraints in Model-Based Tracking

One of the advantages of model-based tracking in general is that objective functions are inherently modular. Given the probabilistic interpretation of an objective function, new terms can be added and balanced against existing terms according to the relative uncertainty in the observations that inform each. We saw one example in the previous chapter with the introduction of the observation SDF. In this chapter, we'll explore how domain-specific knowledge and observations can be used to craft new error terms which can be incorporated into the DART framework to improve tracking performance.

The domain for this exploration is robot in-hand manipulation. We'll assume that there are two robot hands with articulated fingers, and that the robot will be tasked with manipulating objects using one or both hands. We'll also assume that the robot has a head-mounted RGB-D camera. Unlike the experiments in the previous chapter, there are now multiple models to be tracked simultaneously. This makes the task of estimating accurate states more challenging, not only due to the increased dimensionality of the state space but also because the models significantly occlude each other during manipulation (c.f. Figure 3.1). However, we also have access to new information via the robot's sensors which can be used to improve tracking performance to meet the challenge.

In order for the robot to successfully manipulate objects using a model-based planner, the position and orientation of the objects relative to the manipulator and the current values of all manipulator joint angles are needed. A common solution is to visually track the object within the frame of reference of some camera. The object pose is then

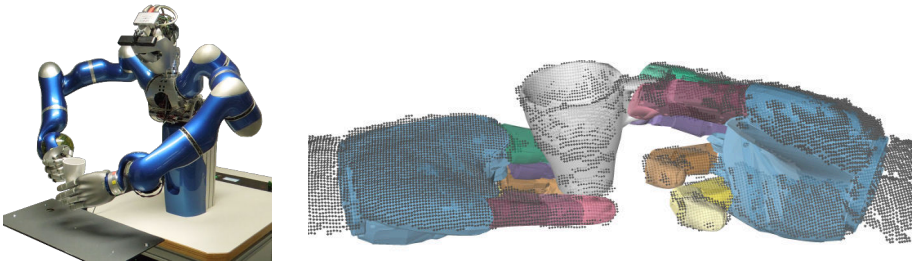


FIGURE 3.1: The SpaceJustin robot is shown at left performing a two-hand manipulation of a coffee mug. At right is the robot's point of view, showing the observed point cloud superimposed on the models of the robot hands and the mug according to the current state estimate. Note that multiple components of the left hand are not observed at all due to occlusion by the mug.

related to the hand pose through a transform from the camera frame of reference to the robot base frame of reference followed by a transform from the robot base frame of reference to the hand frame of reference (Morales et al., 2006; Hertkorn et al., 2013). The latter transformation depends on a calibrated proprioceptive system, while the former depends on a an extrinsic camera calibration. Any small errors in these calibrations can easily add up to large errors in the relative hand-to-object pose, potentially leading to failure of intended manipulations. Knowledge of manipulator joint angles can also be inaccurate for tendon-driven or uncalibrated robots, further complicating grasp planning.

Also key to successful manipulation is the stability of the object pose estimate. If a manipulation is planned based on currently estimated poses, but the estimate changes significantly in the intervening time before the manipulation is executed, the plan may no longer be valid. A good vision system for manipulation must therefore extract stable and accurate estimates even from noisy input data.

We approach the problems caused by calibration errors by tracking both the object and the robot hands in the camera frame of reference. Both poses are then subject to the same intrinsic calibration errors and are already in the same frame of reference, removing the need for extrinsic calibration altogether, and allowing for more accurate relative pose estimation (we also saw this benefit in section 2.4.3).

The issue of estimate stability is handled using a switching model, which combines several frames of data to converge to an accurate and more stable object pose estimate when the object is stationary, either relative to the camera (while on the table) or relative to one of the hands (while firmly grasped). Our switching model requires an automated method for detecting when an object enters and exits these stable states in order to remain robust to unexpected motion of the object. We found that the dense visual error term provides a strong signal for this detection.

We furthermore improve the robustness of tracking for the domain of in-hand manipulation by extending the DART tracking framework described in the previous chapter, adding physics-based terms to the objective function. Physical laws are a useful tool for inferring poses, as it is known with certainty that the state of the system must obey these laws. We'll also show how our signed distance function representation of the tracked models lends itself easily to computing and minimizing this augmented objective function. In particular, in this chapter we incorporate terms which penalize object interpenetration and disagreement between touch sensor feedback and physical contact between objects.

3.1 In-Hand Object State Estimation Overview

In related work, the poses of manipulators and manipulation targets are generally estimated using either physical information, visual information, information from tactile sensors, or some combination of the three. This section gives a brief overview of some of these techniques.

3.1.1 Physics-based tracking

Lowrey et al., (2014) use a physics engine to estimate the state of a walking robot based on a variety of sensors within the robot as well as a fast and highly accurate external

marker-based tracking system. While walking, the robot's feet repeatedly make contact with the ground, and as such the simulator is required to reason about contact forces. To reduce noise, they recompute estimates in a sliding window whenever new data has arrived.

3.1.2 Vision-based tracking

Many techniques used in object and robot tracking systems rely on distinctive keypoints in the image data. Azad et al., (2011) propose a method for tracking (single, rigid) 3D models by instead rendering their edges and computing the overlap between the image edges. Ulrich, Wiedemann, and Steger, (2009) similarly match 3D CAD models in monocular images. In contrast, we use depth sensor data to solve for model poses directly in the 3D space. As one part of the 3D visual tracking, we take advantage of the implicitly represented difference between free and occluded space present in the depth map. Earlier image-based approaches to tracking articulated models did this indirectly with background subtraction and by using multiple cameras to obtain a sense of the 3D layout (Bandouch, Jenkins, and Beetz, 2012). Klingensmith et al., (2013) presented a visual servoing system using an articulated iterative closest point (ICP) variant to track a robot arm, demonstrating that using visual feedback in the control of a robot enables greater robustness to calibration errors. Our work demonstrates the value of visual feedback for tracking manipulated objects in addition to the robot itself.

Krainin et al., (2011) also use articulated ICP, augmented with sparse feature matching and dense color information to track a robot arm and a previously unknown object it has grasped, and to simultaneously build a model of the object. Our articulated ICP variant, accelerated by signed distance function look-ups and a GPU implementation, allows us to achieve real-time performance while additionally reasoning about physics-based constraints. Schulman et al., (2013) use color and depth observations to induce 'observation forces' on a deformable object model and are thereby able to use a physics simulator to find the low-energy state which has the least disagreement with the observation. This work has been developed specifically for deformable objects and it is not clear how it would perform in our setting, which requires much faster update rates and the incorporation of additional constraints.

There are also a number of discriminative learning-based techniques for object pose estimation and tracking. While these are capable of providing rough pose estimates with little to no prior, in general current learning-based approaches do not provide state estimates with sufficient accuracy for planning manipulations.

3.1.3 Contact-sensing-based tracking

Haidacher and Hirzinger, (2003) presented 'a blind man's approach to grasping' in which they search through a set of possible pairings of fingers and object faces to find the pose of an object using only contact detection on the fingers. Koval et al., (2013) propose a particle filter that tracks manipulated objects using only tactile information by sampling particles from a manifold of state space that respects contact constraints. However, it seems unlikely that either of these approaches could provide accuracy commensurate with that provided by a dense visual data term minimized by gradient descent, unless a very large number of particles is used. Furthermore, manipulation of

objects involves states in which the hand is not in contact, in which case contact-only methods can provide no information about the location of the object.

3.1.4 Combined manipulated object tracking

Zhang and Trinkle, (2012) presented an offline particle filter approach that combines visual information, physics, and tactile feedback to track manipulated objects. While they also focus on tracking through occlusion induced by manipulators, the slow update rate makes the approach less applicable to real-time manipulation scenarios. Chalon, Reinecke, and Pfanne, (2013), rather than tracking visually through occlusions, use a vision system to initialize the object pose and then rely on physics and potentially contact information to track a manipulation with a particle filter. As they use the grasp matrix to update the state of the particle filter, manipulation of objects without grasping is not considered. Bimbo et al., (2013) also approach the occlusion problem by fusing tactile information with visual information, but they do not take intersections between object and model into account.

3.2 Extending the DART Objective Function with Physical Constraints

The robot hand and object tracker is based on DART, which was presented in the previous chapter. As a brief review, DART takes as input a depth map, D, and tries to estimate a vector θ describing the tracked state as depicted in that depth frame. This state vector includes the position, orientation, and articulation of all models, and in this chapter also includes the location of contact points between models (described in more detail in section 3.2.2). State estimation is done iteratively using gradient descent. The full error function to be minimized is as follows:

$$\begin{aligned} E(\theta; D) =& E_{\text{mod}}(\theta; D) + \lambda_{\text{obs}} E_{\text{obs}}(\theta; D) \\ &+ \lambda_{\text{int}} E_{\text{int}}(\theta) + \lambda_{\text{con}} E_{\text{con}}(\theta) \,, \end{aligned} \tag{3.1}$$

where E_{mod} and E_{obs} represent the error terms in the original DART framework, measuring the error induced by observed points in the model SDF and by predicted model points in the observation SDF, respectively. We will now describe the final two terms, which are contributions of this chapter.

3.2.1 Intersection Term

Based on the simplest physical principles, we know a priori that the union of physical space occupied by any pair of rigid bodies must be empty. The visual terms alone should be sufficient to ensure that the pose estimates satisfy this condition under full visibility, but in times of heavy occlusion, as are common in manipulation scenarios, this physical constraint becomes a useful source of information for estimating otherwise unconstrained degrees of freedom[1].

[1]For example, in Figure 3.1, we cannot tell exactly where the fingertips of the left hand are from this single observation, but we can infer where they *aren't* based on the 3D object models of the hands and mug and the fact that they are both solid rigid objects.

Suppose there are two rigid bodies, A and B, represented implicitly with continuous signed distance functions $f_a(x, y, z)$ and $f_b(x, y, z)$, which give for every point in 3D space the shortest distance to the surface of the respective rigid body. Noting that SDFs are *negative* inside a body, a natural way to penalize the intersection of these two bodies would be as follows:

$$\iiint \min\left(0,\ f_a(x,y,z)\right) \min\left(0,\ f_b(x,y,z)\right) \mathrm{d}x\, \mathrm{d}y\, \mathrm{d}z\,. \tag{3.2}$$

However, a triple integral over a discrete representation of a signed distance function would be quite expensive. We then note that taking interior parts of both models into account simultaneously is unnecessary, as no point on the interior of B can be inside of A without some point on the surface of B having penetrated first. We can then simplify the triple integral by replacing it with two surface integrals:

$$\oiint \min\left(0,\ f_a^2(x,y,z)\right) \mathrm{d}S_B + \oiint \min\left(0,\ f_b^2(x,y,z)\right) \mathrm{d}S_A\,, \tag{3.3}$$

where S_A and S_B are the surfaces of the two rigid bodies. Finally, we can discretize this surface integral by considering only a finite set of points on the surface of all rigid bodies. Thus, as a pre-processing step we store a collection of points X^m for each model m in our set of tracked models, M, such that all $x \in X^m$ lie on the surface of model m. This is done by sampling points on the mesh faces uniformly by surface area. The points are stored in the frame of reference of the rigid body from which they were generated, and then transformed into the global frame via the kinematic chain of the articulated model[2]. The error induced by points on model m_a penetrating model m_b is given by:

$$e_{\text{int}}^{m_a,m_b}(\theta) = \sum_{\mathbf{x_i} \in X^{m_a}} \min\left(0,\ SDF_{m_b}\left(T_{m_b,f_i}(\theta) * \mathbf{x_i}\right)\right)^2, \tag{3.4}$$

where f_i is the frame of reference in which point x_i moves rigidly, and T_{m_b,f_i} is the transform from that local frame of reference to the frame of reference of model m_b (equivalent to root frame 0 in Chapter 2, except that there are now multiple model roots). This transform is defined by the kinematic chain and the current parameter estimates θ according to

$$T_{m_b,f_i}(\theta) = T_{m_b,c}(\theta) * T_{c,m_a}(\theta) * T_{m_a,f_i}(\theta)\,. \tag{3.5}$$

In words, a point is placed in the frame of reference of a target model by chaining transformations through the root frame of the source model to the camera frame and then to the target model frame of reference. The full intersection term we then minimize is:

$$E_{\text{int}}(\theta) = \sum_{m_a \in M} \sum_{m_b \in M} e_{\text{int}}^{m_a,m_b}(\theta)\,. \tag{3.6}$$

Note that equation 3.4 is directional, so we consider in equation 3.6 both the possibility of points from m_a intersecting m_b and vice versa. We also consider points from m_a self-intersecting other parts of model m_a, which can happen the model is articulated.

[2]This set of points plays a similar role to the predicted point cloud described in section 2.2.4, except in this case we do not use a rendering to generate the point cloud because we are also interested in reasoning about model surfaces which are not visible.

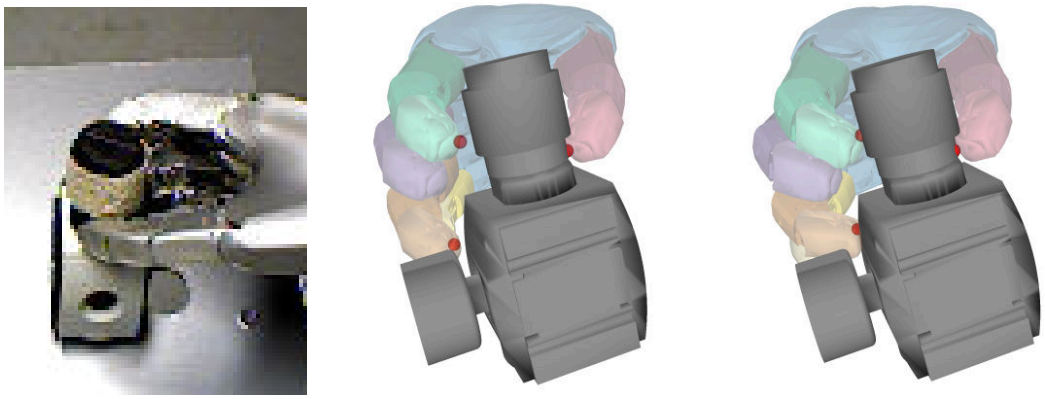


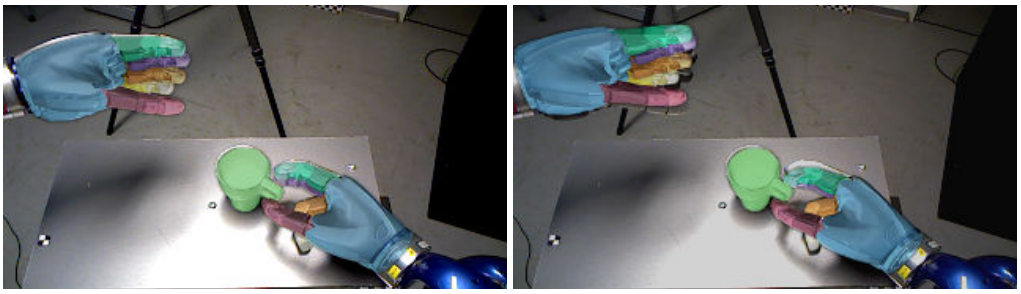
FIGURE 3.2: (A) A grasp in which contacting fingers are occluded is executed. (B) and (C) show a rendering of this grasp from another viewpoint, with estimated contact points for fingers detected to be in contact shown as red spheres. (B) the contacts are first detected via the torque sensors in the hand. The thumb is well observed and thus the state estimate is already more or less consistent with the detected contact. The two opposing fingers are not well observed and the estimate is inconsistent with the detected contact. (C) The estimate of the object pose, hand pose, and the location of the contact points is updated by minimizing equation 3.7 such that the identified fingers touch the object.

This formulation is almost identical to the error induced by observed points in the model SDF as presented in the previous chapter, except for the truncation of the SDF to interior regions. We are thus able to compute first order derivatives for the error induced by intersecting points exactly as in DART (c.f. section 2.2.4).

3.2.2 Contact Term

While the intersection term applies generally to any rigid body, many robot hands can provide additional helpful information by sensing when contact has been made with other surfaces. This is particularly useful in grasping scenarios, as many natural grasps involve the placement of fingers behind the object, where they are visually occluded.

Given the location of contact on the finger, perhaps from a high-resolution tactile sensor, the goal would be to simply minimize the squared distance from the contact point on the finger to the surface of the object, such that the pose estimate will reflect that contact is in fact being made. However, we are interested in a wider range of systems in which the existence of contact can be detected, perhaps via torque sensors, but not the location of the contact point; this location then becomes a hidden variable that needs to be estimated. While this location is minimally represented as a point on the two-dimensional manifold of the finger surface, we instead chose to simplify the optimization objective, and overparameterize the variable by representing the contact location as a 3D point in the frame of reference of the finger. We simply project the estimated point back onto the finger surface after every step of the gradient descent.



(A) Estimated State (B) Reported State

FIGURE 3.3: (A) An overlay of the state estimate on an example frame. (B) The right hand is shown according to its pose estimate, while the left hand is positioned relative to the right hand according to the forward kinematics and reported joint angles of the left and right arms, and the fingers are shown positioned as reported by the robot. Note the errors that are particularly evident in the left thumb and right index finger.

We define a variable c_i for each finger which takes on a value of 1 if contact was detected on that finger and a value of 0 otherwise. Our contact error term is then:

$$E_{\text{con}}(\theta) = \sum_i c_i \text{SDF}_{\text{obj}}(T_{obj,i}(\theta) * \mathbf{p_i}(\theta))^2 \,, \tag{3.7}$$

where $\mathbf{p}_i(\theta)$ is the estimate of the contact point on finger i and SDF_{obj} is the signed distance function representation of the object being manipulated. Once again, we have an error based on the look-up of an implicit point-to-surface distance of a point defined in a local frame of reference. The derivatives of this term with respect to the hand and object poses are thus computed as in the intersection or observation-to-model error terms. For the derivatives with respect to each contact location, we have:

$$\frac{\partial}{\partial \mathbf{p}_i} E_{\text{con}}(\theta) = c_i \nabla \text{SDF}_{\text{obj}}(T_{obj,i}(\theta) * \mathbf{p}_i(\theta)) \,, \tag{3.8}$$

which is simply the gradient of the object SDF evaluated at the contact point. The effect of this term can be seen in Figure 3.2.

3.2.3 Parameterization

When tracking robots that are capable of providing proprioceptive feedback, we have some prior knowledge over the pose. However, the combination of calibration errors in the proprioceptive system and the camera means that the joint angles reported by a robot cannot be trusted exactly, as demonstrated in Figure 3.3. We therefore use the reported joint angles, but do not rely on them entirely.

One approach for incorporating proprioceptive information into a visual, gradient-descent-based tracking framework is to treat reported joint angles as a prior in the Bayesian sense (Krainin et al., 2011). However, with our focus on occlusions, we found this approach to be suboptimal, due to a 'snap-back effect': when any degree of freedom

becomes unobserved, it simply reverts back to the reported prior regardless of whether that prior had matched observed information in recent frames.

We instead took the approach of Klingensmith et al., (2013), choosing parameters that represent relative offsets from the reported joint angles rather than absolute angles. That is, at each time step, instead of finding the absolute joint angles θ that minimize the error function, we optimize over relative joint angles δ that minimize the error when added to reported joint angles $\hat{\theta}$:

$$\delta^* = \arg\min_{\delta} E(\hat{\theta} + \delta; D) \,, \tag{3.9}$$

where $E(\theta; D)$ is the error function of equation 3.1. This parameterization has some nice properties, first and foremost being that if we assume the offset between the actual and reported joint angles is mostly constant, we can highly regularize our gradient descent steps to ensure the relative values change slowly and are less sensitive to sensor noise. As the amount of regularization approaches infinity, the tracker will follow the joint angles exactly, while a high but finite value allows for slow and steady adjustments to accommodate bias inherent to certain camera views and errors in system calibration (or lack thereof), and also allows the tracker to function when the offset is not actually constant but varies somewhat over time and with different configurations.

3.2.4 Switching Model

While we have a relatively strong signal as to where the hands are through the forward kinematics of the robot arms, we are not so lucky when it comes to the object position. To produce stable estimates of the relative transform between hand and object as needed for grasp planning, we follow the work of Krainin et al., (2011) in identifying three possible states of the object:

1. The object is at rest on a surface.
2. The object is in an intermediate unstable state.
3. The object is stably grasped by a hand.

However, while Krainin et al. assume that they know the state based on the actions taken by the robot (i.e. the object is on the surface when the robot places it on the surface and stably grasped when the robot has performed a grasp), we make no such assumptions. Instead, we present an automatic switching model that detects state transitions based on observations of the model, making our tracker robust to failed grasps and slippage or unintended dropping of grasped objects, and also enables manipulations that don't involve grasping at all, such as pushing.

In order to detect state transitions, we first define our measure of the 'visual error' of a particular estimate as the average distance from all observed object points to the object model SDF surface, plus the average distance from all predicted object model points to the nearest unobserved space in the observation SDF. So, if O observed points are associated with the object and the predicted depth map has P object points, this metric will be $\frac{E_{\text{mod}}}{O} + \frac{E_{\text{obs}}}{P}$. The intuition here is that when the estimate is accurate, all predicted object points and all observed points associated to the object should have low error, and when the object moves, there will be at least some predicted and observed

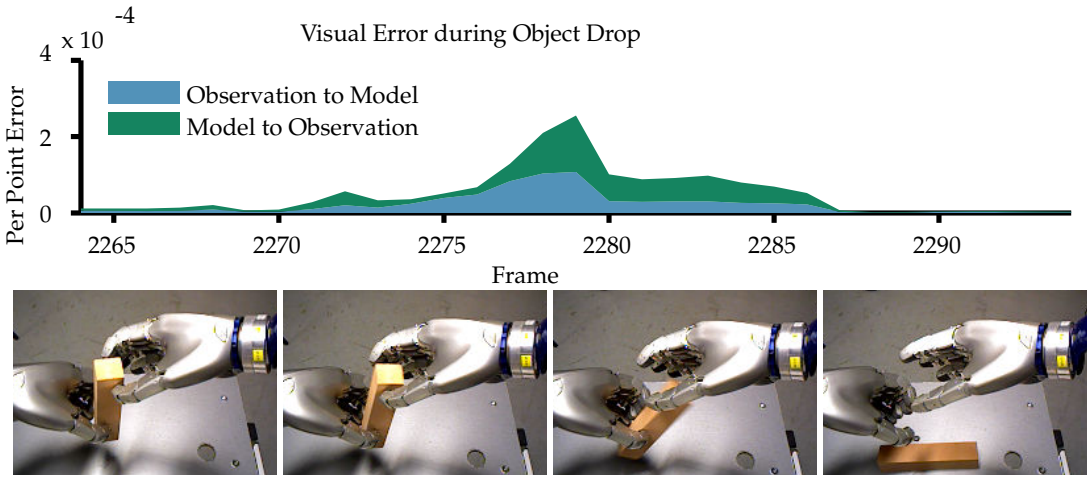


FIGURE 3.4: Visual error for a sequence of tracked object pose estimates, before optimization. The object begins in a stable grasp state, transitions to the intermediate state (frame 2267) due to the high error indicating the object is not where it is expected to be based on proprioceptive updates composed on the last frame's estimate, and finally enters the stable rest state (frame 2287).

points with high error, as shown in Figure 3.4. This value is computed after updating the pose estimate according to the latest proprioceptive feedback from the robot but *before* optimizing the objective function.

The object is always initialized in state 2, and transitions to one of the two stable states once the pre-optimization error metric falls below a predefined threshold. If the fingers are detecting contact when this happens, the object transitions to state 3, and otherwise transitions to state 1. The threshold is set just above the visual error typically observed when the object is at rest on a surface, which depends on the noise model of the sensor in use but is easily determined empirically.

The estimation of object state is useful in that we can assume that the object is stationary in the stable states — stationary relative to the world frame in state 1 and stationary relative to the root frame of the grasping hand in state 3. Based on this assumption, we could collect a series of frames and solve for the 6 parameters which minimize the sum of the error in all frames. We approximate this in a real-time and online manner by keeping a running estimate of the amount of information we have about each of the 6 degrees of freedom. We first note that our Hessian approximation $H = J^T J$ is the inverse of the covariance matrix, and therefore that entries along the diagonal are inversely proportional to the (squared) uncertainty of the corresponding variables according to the local linearization, i.e. $H_{ii} = 1/\sigma_i^2$. We therefore store a running sum, ι, of the inverse uncertainty of the object pose parameters in all the frames since entering a stable state. If the object is estimated to be in a stable state at time step t, we set:

$$\iota^{t+1} := \min(\iota^t + \alpha \text{diag}(H^t),\ \iota_{\max}) \,, \tag{3.10}$$

where α is an 'accumulation rate' parameter and $\iota_{\max}$ limits the amount of regularization that can be applied to any particular degree of freedom. The exact value of α is not critical and is generally set anywhere between 0.2 and 0.8 in our experiments. The

parameter $\iota_{\max}$ actually has a unit, namely the inverse square of the unit of the corresponding degree of freedom, and is set accordingly. Then, we solve the following regularized normal equations for each Gauss-Newton step:

$$\Delta\theta^t = -(J^\top J + \iota^t)^{-1} J^\top r \,. \tag{3.11}$$

As a result, after the object has remained in a stable state for a number of frames, the estimate will change more slowly, depending on the amount of uncertainty in all frames, and should thus converge over time to a stable estimate of the true object pose and no longer be influenced by noise on a frame-to-frame basis. Crucially, each degree of freedom is regularized differently depending on our certainty of its state. This allows us to, for example, converge to stable estimates for observed fingers while maintaining uncertainty over the pose of unobserved fingers such that the estimate can adapt rapidly when they are ultimately observed. Once we have detected a deviation from the stable state as previously described, we assume previous information is no longer valid and set $\iota = 0$ until we have again entered a stable state.

3.2.5 Data Association with Multiple Models

When tracking multiple models with DART, one might be tempted to simply instantiate a separate DART tracker for each object. In fact, there are reasons this might be desirable beyond simplicity, such as increased parallelization. However, there is one important step in which joint reasoning over the models is critical to robust tracking: the data association step (i.e. equation 2.8). With multiple models and separate tracking instances, the same pixel might get associated with one model for one instance and with another model in another instance. This is particularly likely in the scenario we are interested in, as the robot hands and the object are expected to be in close proximity. If too many observations of points on the hand are incorrectly associated with the object (or vice versa) it can catastrophically corrupt the pose estimates.

We therefore perform joint data association taking all models into account, meaning each pixel is only associated with one model. This allows each model to 'explain away' the points that it best predicts such that they do not interfere with the tracking of other models. After data association, the points assigned to different models can be processed in parallel.

The need to do joint data association is not much of a problem for the objects and hands as we are interested in tracking them and therefore are required to estimate the data association. However, we also have problems with the object becoming confused with the table surface if there is no way to 'explain away' the table. We therefore *also* track the table plane in three degrees of freedom and subtract the points associated with the table before computing data association[3]. This demonstrates the problems caused by ambiguous data association (for more on this topic, see Chapter 5). An alternative would be an appearance model that is capable of distinguishing between objects at a pixel level based on the RGB image. We'll return to this idea in Chapters 6 and 7.

[3]If the support surface were not planar we could model it using, e.g., KinectFusion (Newcombe et al., 2011), and subtract it all the same.

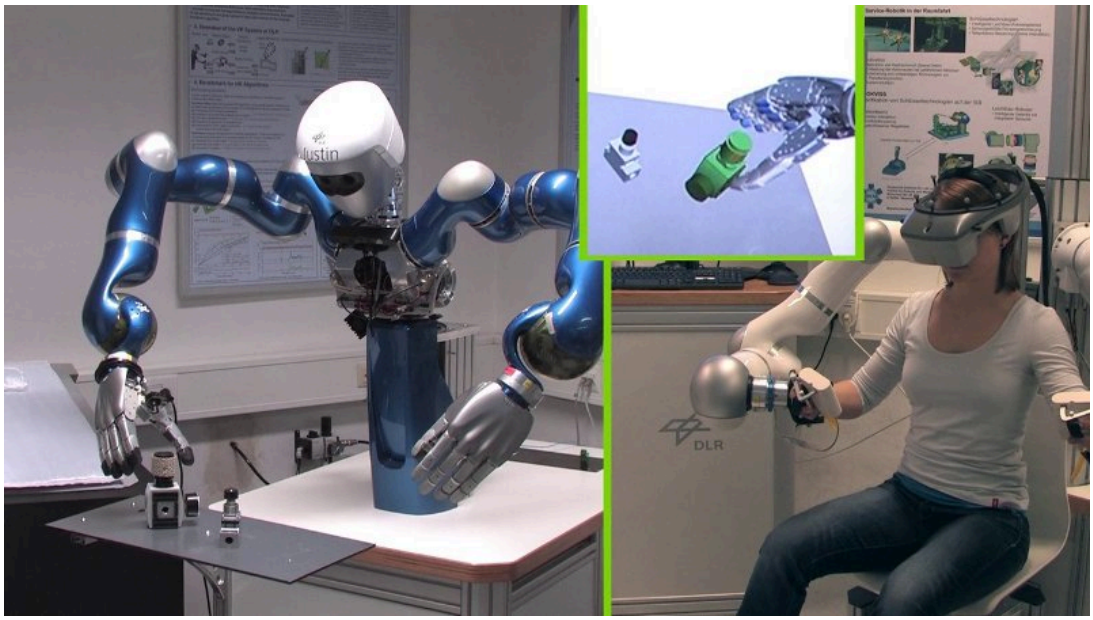


FIGURE 3.5: Semi-autonomous grasping using DLR's telepresence system. It consists of the remote robot SpaceJustin (on the left) and the human-machine interface HUG (on the right). The visual assistance is displayed using the head-mounted display.

3.3 Experiments

As a platform for the proposed method, we use a shared autonomy system consisting of a telepresence robot equipped with online grasp planning capabilities (Hertkorn et al., 2013). However, the only system-dependent assumption made is that the hands are capable of estimating whether each finger is in contact with an object, and that the system can report internal estimates of joint states. The tracking approach should apply to any system that satisfies these requirements.

Grasps for two five-finger hands are planned online according to the current hand-to-object relative pose. This allows for planning of grasps in unexpected situations as the grasps are not restricted to those available in a grasp database. The shared autonomy setup poses additional challenges to the tracking system, as the human operator can move the arms of the robot freely (in contrast to executing a pre-planned motion), potentially bumping the objects or otherwise changing the hand-to-object relative pose. In such cases, an updated pose estimate for the object is needed to plan a new grasp to be executed. Following previous work (Hertkorn et al., 2013), we use the static scene analysis as an initial guess for the presented tracking method, which, being a local gradient descent method, needs a rough initialization. We show the benefits of the physical constraints using a synthetic manipulation as a baseline, and in the accompanying video (linked at the end of the chapter), we also present tracking estimates for a number of challenging manipulations for qualitative analysis. All experiments were run with the error function weighted according to $\lambda_{\text{mod}} = 4$, $\lambda_{\text{int}} = 10$, and $\lambda_{\text{con}} = 25$.

3.3.1 System Details

The system consists of a multimodal human machine interface HUG (Hulin et al., 2011) and the robot SpaceJustin, which is a modified version of humanoid robot Justin, both developed at the Deutsches Zentrum für Luft- und Raumfahrt (DLR) (Borst et al., 2007). Both components of the system are shown in Figure 3.5. SpaceJustin has 17 actuated degrees of freedom (DoF) for its torso, head, and arms, and interacts with the environment using two DLR-HIT Hands II which have 15 actuated degrees of freedom each[4] (Liu et al., 2008). An Asus Xtion is mounted on the head and is used to track the hands, fingers, and the object. The robot arms of HUG and SpaceJustin are coupled in Cartesian space which allows the operator to control the movements of SpaceJustin's arms and experience realistic force feedback. A one degree of freedom hand interface is used to trigger the grasping command once a stable grasp is planned. The operator receives visual feedback by wearing a head-mounted display (HMD, NVisorSX60 from NVIS) showing the remote and estimated environment in 3D. In order to allow a high degree of immersion, the operator's head movements are also tracked, enabling control of the movement of SpaceJustin's head (and thus the position of the camera as well).

The robot system reports its current joint angles (47 in total) to the tracking system at a rate of 1 kHz. Additionally, the fingers provide a vector of binary inputs to the tracking system indicating whether each finger is currently in contact with a foreign object, as estimated by the torque sensors integrated in every finger. The Asus Xtion provides depth information at a rate of 30 frames per second. Tracking and grasp planning run at this update rate. The difference between estimated joint angles and measured joint angles as calculated by the tracking system is sent back and taken into account by the hand controller to correct for the errors in forward kinematics.

3.3.2 Synthetic Experiment

To show the contribution of newly introduced components of the tracking system, we generated a synthetic manipulation of a small sphere using the Bullet physics engine[5] and a corresponding sequence of depth maps. The trajectory from the physics simulation gave us a baseline for comparison, although it should be noted that the simulation was not perfectly realistic. The rendered depth maps were corrupted with Gaussian noise with a standard deviation of 2 mm, sampled at a quarter of the depth map resolution such that the noise in neighboring pixels is correlated, and discretized into 1 mm bins as in consumer depth sensors. The known true odometry is corrupted by a zero-mean Gaussian walk.

We compare in Figure 3.6 the tracking performance of DART with and without the additional physical constraints on this challenging sequence, where the error metric is

[4]Each finger actually has 4 degrees of freedom: one each for flexion/extension of the three links, and one for the adduction/abduction of the base link. This brings the total degrees of freedom to 20, the same as the DART hand model from Chapter 2 excluding the 6 degrees of freedom for the wrist pose. However, the flexion/extension of the last two links of each finger are mechanically coupled such that the angle of rotation of their joints is the same. We made a simple extension to the DART tracking framework to take this into account. Poses are represented in the lower-dimensional manifold of the realizable pose space, and can easily be projected to the higher-dimensional ambient pose space. When computing gradients we also take this mapping into account and combine gradients for links which are constrained to have the same joint angle.

[5]https://bulletphysics.org/

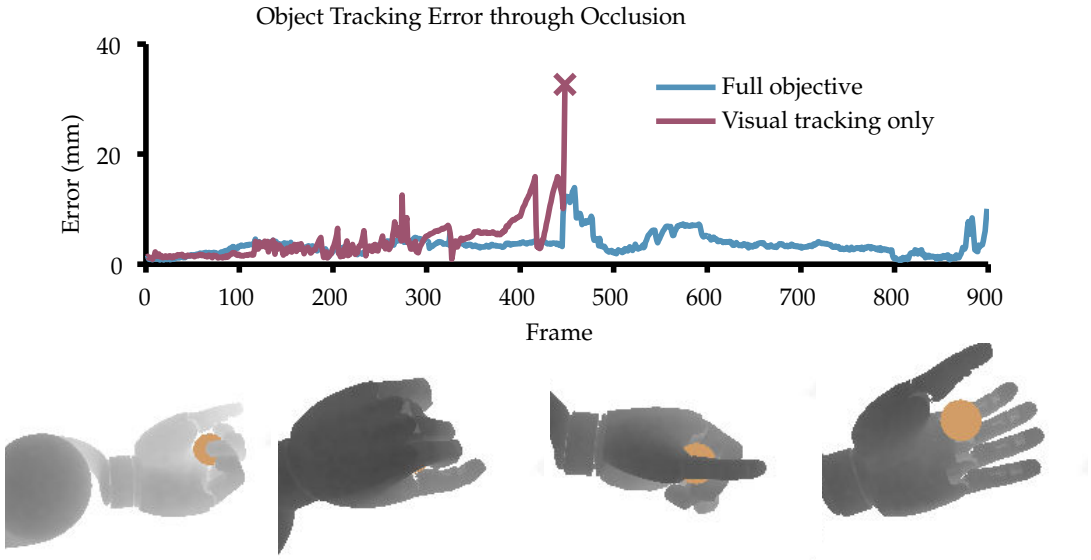


FIGURE 3.6: Above is a plot of the error in 3D of the estimate of the position of a sphere being manipulated by a robot hand in a synthetic sequence. The row of images below show a selection of frames from the sequence with the sphere highlighted in orange. Without physical constraints, tracking fails when the sphere becomes hidden.

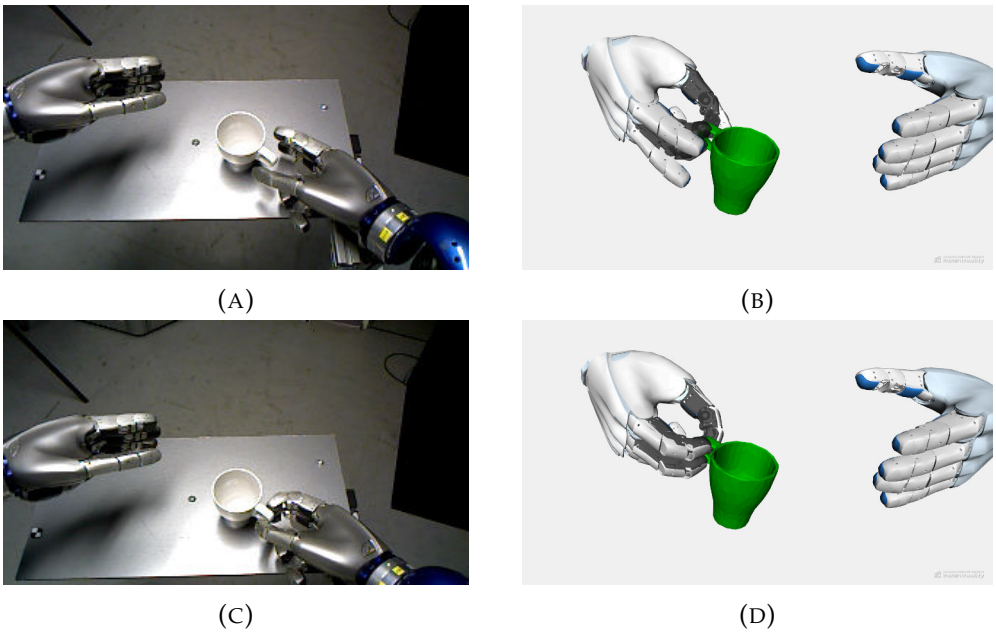
given by the Euclidean distance between the predicted and ground truth sphere center. As shown, the additional terms in the objective function lead to more stable estimates through the extreme occlusions in this sequence. Furthermore, once the sphere becomes nearly entirely invisible to the camera, the vision-only system experiences a tracking failure from which it does not recover. With physical constraints, tracking is successfully maintained through the entire sequence, with a mean error of 3.5 mm.

3.3.3 Real Experiments

Due to the aforementioned difficulties in performing quantitative evaluation of markerless tracking methods, we also qualitatively demonstrate a number of manipulations we were able to perform due to the accurate and fast pose estimates provided by the presented tracking system, all of which may not even have been possible otherwise. The experiments are conducted using the same method for all objects and situations; no tuning of parameters is needed either in the tracking or the grasp planning.

The first challenging manipulation is the execution of a grasp with a narrow margin of error, which demands a high degree of accuracy, as any deviation in the estimated joint angles can cause the fingers to miss the small intended targets. Figure 3.7 shows one such challenging grasp, in which three fingers are used to grasp a small handle on a coffee mug. Not only is the target small, but the mug surface is slippery and the center of mass is far from the contact points, all of which contribute to the difficulty of the grasp. By tracking both the hand and mug in the camera frame of reference, we are able to estimate the relative poses with enough accuracy to successfully complete the grasp.

Another challenge for a tracking system is the movement of the camera, as we want the operator to be able to change her viewpoint to facilitate manipulation. The robot also



(A) (B)

(C) (D)

FIGURE 3.7: A grasp that demands high accuracy. (A) and (C) show the robot's point of view, (B) and (D) show the corresponding estimated joint angles and relative hand-to-object poses. The planned grasp is also shown in dark gray.

provides proprioceptive feedback on the motion of the head, but as is the case with the arms, relying on these estimates and the forward kinematics of the robot would require highly accurate knowledge of the camera position in the head frame of reference, as well as a very high quality intrinsic calibration of the camera. Instead, we use only a very rough initial estimate of the extrinsic calibration to predict the motion of the hands and object relative to the camera during neck motion, and then rely on the visual tracking to account for the errors. An example of our robustness to changing camera viewpoint can be seen in Figure 3.8.

We were mostly interested in bi-manual manipulations, which pose additional challenges to the tracking system. In previous work on the same shared autonomy system (Hertkorn et al., 2013), the object pose was estimated while it lay stationary on the table, not occluded by either hand. When passing the object from one hand to the other, however, the grasp planner needs to know the relative transformation between the grasping hand and an object which is not on the table. Estimating this relative transformation is challenging because the object is occluded by the fingers of the other hand, and could be moving; the previously used pose estimation system was not capable of estimating the pose in these conditions.

Finally, we did not want to make the limiting assumption that every grasp attempted by the operator would be successful, and we did not want to have to restart the tracking system every time an object moved unexpectedly. We show in the accompanying video that while our tracking system is able to use information from multiple frames to stabilize object pose estimates, it is also able to detect unexpected motion and react quickly when the object moves during grasping or drops suddenly from the hand. In these cases, the operator is usually able to immediately begin correcting the failure, as

an accurate estimate of the object pose is still available.

Discussion

We presented the benefit of incorporating proprioceptive information and physical constraints into dense visual articulated model tracking, and applied it to jointly track robot hands and the objects they manipulate. The implicit signed distance function representation of the tracked models allows us to easily detect interpenetration of multiple interacting models, as well as self-intersection of a single model, and to correct these intersections by adding a new term to the visual error function. After introducing a hidden variable for the location of contact between models, we are also able to use our SDF representation to add a fourth term into the error function which favors pose estimates which explain detected contacts. The intersection term is as trivially parallelizable as the visual error terms, and the contact term is processed at essentially no cost on the CPU as all other terms are being processed on the GPU. We are therefore able to use the model pose estimates for planning and control in a bi-manual shared autonomy system, achieving real-time update rates of 30 frames per second while tracking 48 pose parameters (two 21 degree of freedom hands plus a 6 degree of freedom object). We also introduced an automatic switching model that detects stationary states of the object and reacts accordingly, in order to provide tracking estimates that are stable and highly accurate when the object is not moving, yet keep up when the object does move quickly.

In the accompanying video, we show a number of manipulations that were enabled by using the estimates provided by the tracking system to plan and execute grasps. We sought to make the manipulations challenging by executing grasps with little room for error, such as lifting a coffee mug by a small handle, by moving the camera throughout the manipulation without accurate knowledge of the camera position relative to the kinematic tree, and by grasping objects that are already held (and thus occluded) in the other hand. Throughout the course of the manipulations, there were also failed grasps, unintentional drops, and non-grasping manipulations such as pushing the object across the table or using one hand to rotate an object held in the other hand, and we are able to maintain accurate tracking through these situations as well. While we demonstrated all results on tracking manipulations executed by a shared autonomy system, the presented tracking method is equally applicable to fully autonomous systems.

In this chapter and the last, we have explored a generative model-based system for tracking generic articulated objects. We have seen this system applied to a number of applications, including human hand tracking, human body tracking, and the tracking

FIGURE 3.8: Tracking of the hands and object is maintained through changes in camera view. This allows the human operator to reposition the robot head (and thus the camera as well) for a more natural viewpoint on the manipulation.

of multiple robots as they manipulate a number of objects, articulated or otherwise. The wide range of applications demonstrates the portability of the model-based approach, and the ease with which additional domain-specific terms could be added in this chapter demonstrates its modularity. One of the remaining limitations of this system is its reliance on an external estimate to initialize the object pose at the beginning of tracking. In the next chapter, we'll explore one way in which this generative model-based tracking can be leveraged to build a system to provide these estimates.

Additional Resources

- A visual overview of how the additional constraints work in DART can be found in this video: https://youtu.be/sOQK7DuGjAc. It visualizes the optimization of the visual error, the contact error, and the intersection error, in that order.
- A video showing additional experimental results from the SpaceJustin experiments can be found here: https://youtu.be/cUdKxgFgG00.

Chapter 4

Leveraging Model-Based Tracking to Automate Object Pose Labeling

In the previous chapter we detailed a system for tracking an object and a pair of robot hands during manipulation. As discussed in Chapter 3, no initialization was needed for the robot hand poses, as the robot's proprioception could be relied on to provide a reasonable estimate of the poses that was sufficient to begin tracking. This was not the case for the objects, however, which was one of the limitations of the system. Object poses were estimated by a vision system which only worked when the object was alone on a flat surface and not occluded by any other object, including the robot hands. Thus, while we were in most cases able to track objects through complex manipulations, if the system ever loses track of an object it must be returned to the table and left alone in order to resume tracking.

In recent years, the state of the art in object pose estimation from RGB images has advanced considerably (Brachmann et al., 2014; Tekin, Sinha, and Fua, 2017; Xiang et al., 2018; Li et al., 2018). While these works have shown impressive performance on this problem, they have been able to do so only because of large and labeled training sets. These datasets are expensive to acquire and annotate.

In this chapter, we introduce a simple idea to make dataset collection for object pose estimation much cheaper: if a model-based tracking system is capable of tracking object poses in a video sequence, then it can be used to automate the pose labeling process. Even if poses must be manually annotated for the first frame, the effort for this one frame can be used to automatically propagate class and pose labels for potentially thousands of subsequent frames. Furthermore, the manually-provided labels do not have to be accurate as it is only used for initialization, which further speeds up the labeling process.

4.1 The YCB-Video Dataset

The YCB dataset comprises a number of textured mesh models of common objects, physical copies of which can be acquired for benchmarking performance on tasks such as robot grasping and manipulation (Calli et al., 2017). Our effort on semi-automated object pose labeling uses these objects and is therefore called the YCB-Video dataset. To begin, we selected a subset of 21 objects from the YCB dataset. These objects were chosen because they appear well in structured-light depth images, can be moved without deforming, and have been previously reconstructed with a high quality mesh model.

FIGURE 4.1: The subset of the YCB objects used in the YCB-Video dataset

Renderings of the textured models used in the YCB-Video dataset can be seen in Figure 4.1.

4.1.1 Data Collection

For each video sequence, we selected a subset of between three and nine of these objects and placed them in a novel configuration. Then, an RGB-D video was recorded with an Asus Xtion sensor, which was moved so as to image the objects from a variety of viewpoints and distances. Care was also taken to move the sensor such that objects occluded each other in some frames. Example frames from a number of video sequences are shown in Figure 4.2.

4.1.2 Data Processing

The first step of processing each sequence was to manually identify the set of objects that appears in the video. Then, a manual pose initialization was provided via a 3D user interface, which involved clicking four points on the object model and the corresponding four points in the point cloud. These corresponding point pairs were used to align the model with the observation via the Procrustes algorithm (Gower, Dijksterhuis, et al., 2004).

A modified DART tracker is used to estimate poses of the objects, so the next step is to compute signed distance functions as described in section 2.2.1. As in section 3.2.5, when tracking multiple models it is important that data association be computed jointly, taking all objects in the scene into consideration such that each pixel is only associated with one rigid component of one model. This was especially important in the creation of the YCB-Video dataset as the objects are often very close or even stacked on top of each other. If a separate DART tracker were used for each object, points from neighboring

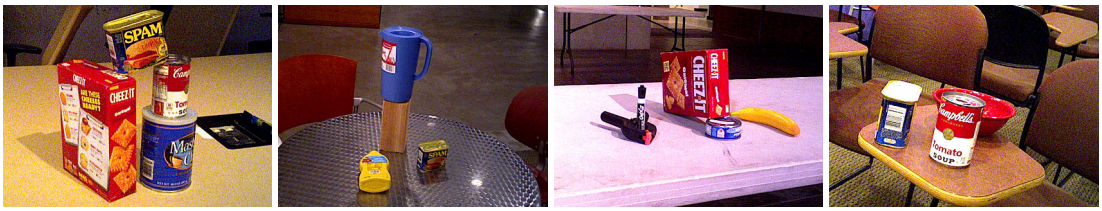


FIGURE 4.2: A sampling of frames from the YCB-Video dataset.

objects would be incorrectly associated. Joint data association, on the other hand, allows the neighboring object model to 'explain away' these points. For the same reason, we also track the planar surface on which the objects rest, again as in section 3.2.5. This means we assume planar support for all objects, but we still allow for objects resting on other objects, and the assumption could be relaxed by simply using dense SLAM techniques to model any non-planar surfaces beforehand.

For standard DART tracking with N rigid objects, the goal is to track object poses $T_{c,i}$ for $i \in \{0, ...N\}$, and this is in fact what we do for the first frame of each sequence in the YCB-Video dataset. However, we then make the further assumption that the objects do not move relative to each other for the duration of the sequence. We therefore instantiate a 'world' frame of reference w, defined as the pose of the camera in the first frame, i.e.

$$T^0_{c,w} = I \, . \tag{4.1}$$

We can then define the object poses according to this world frame, as in

$$T^i_{c,o} = T^i_{c,w} T_{w,o} \, , \tag{4.2}$$

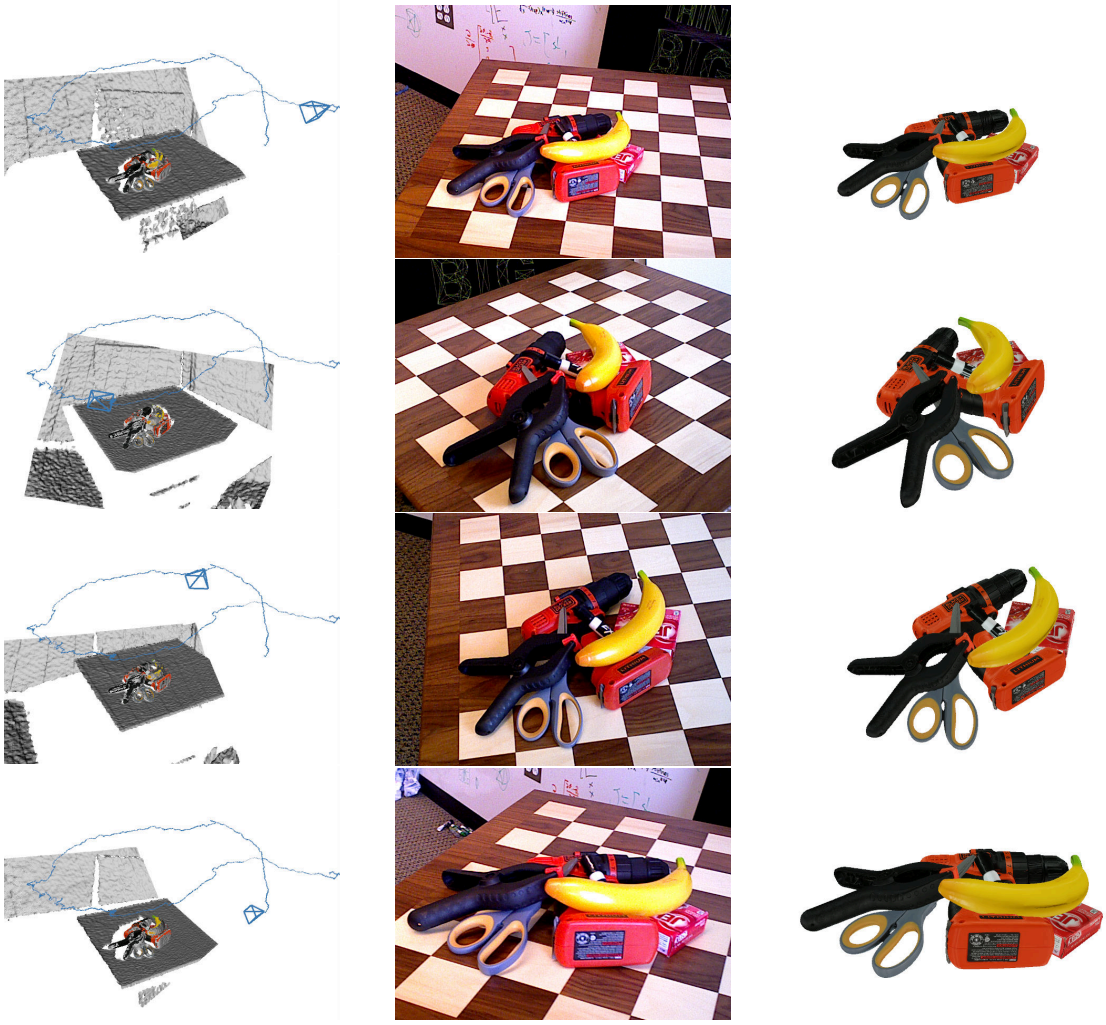
where o is the index of the object and i is the frame index. Note that there is no frame index in the object-to-world transform as this is assumed static, and that composing the definition in equation 4.1 with equation 4.2 we have

$$T^0_{c,o} = T_{w,o} \, . \tag{4.3}$$

After initialization of the object-to-world poses $T_{w,o}$, we temporarily fix the object poses and use all objects in the scene to compute an initial estimate of the camera trajectory. To do this, we essentially treat the world frame as a root frame to which all models are rigidly attached. We can then compute derivatives exactly as in section 2.2.3, with the result being that gradients from all objects are combined to estimate updates to the camera pose $T^i_{c,w}$ for each frame i.

As we are tracking the camera pose relative to *all* objects, it is not necessary that each individual object is observed so as to fully constrain all six degrees of freedom. If, for example, only one planar face of an object is observed in a particular frame, the camera pose can still be resolved using the other objects in the scene. Treating the objects as a static configuration is also of immense help when objects become heavily occluded, as object poses tend to become more ambiguous as less of the object is observed.

Due to noise and complex sensor-dependent distortions in the depth map, the object-to-world poses estimated in the first frame are likely to be biased. To account for this, we perform a few iterations of gradient descent to jointly optimize the object-to-world pose



(A) 3D Scene (B) Color Frame (C) Pose Estimates

FIGURE 4.3: A visualization of one of the sequences from the YCB Video dataset. Each row represents one frame from the sequence. Column (A) shows the 3D scene, including the object models in their estimated world pose, the back-projected depth map, the camera trajectory, and a view frustum for the frame. Column (B) shows the frame captured by the RGB sensor. Column (C) shows the object models rendered according to the estimated object-to-camera pose.

for each object and the camera pose for each frame. This allows us to both improve the estimate of the object configuration and improve the estimate of the camera trajectory. Finally, equation 4.2 is applied to extract the final object-to-camera pose estimate for each object for each frame, ready to be used for learning. Figure 4.3 visualizes the final state estimate for one sequence in the dataset.

4.1.3 Symmetric Pose Resolution

Because only the depth frames were used to estimate object poses, a few of the objects that are geometrically symmetric have pose estimates that are correct up to a transformation within the symmetry group. For some objects, such as the wood block, it is extremely difficult even for a human to resolve a consistent pose, while other objects such as the soup can have enough texture to disambiguate the pose. This also had to be done manually, which for rotational symmetry with one degree of freedom required only two clicks per rotationally symmetric object per video: one on the textured object model and another on the corresponding point in any of the color frames. The pose was then modified to minimize the distance between these two points while remaining within the symmetry group. The modification is made to the object-to-world pose $T_{w,o}$ and is therefore automatically propagated to every other frame in the sequence.

Discussion

In this chapter, we used the same model-based tracking technique discussed in Chapters 2 and 3 to automate most of an object pose labeling pipeline. We were thus able to leverage only a few thousand clicks by a human to label over 130,000 video frames (in comparison, the LINEMOD dataset by Hinterstoisser et al., (2011) contains about 18,000 video frames). Furthermore, because each video contains on average 4.47 objects, there are almost 600,000 total object pose labels.

The YCB-Video dataset has limitations. For one, while the number of frames is large, the variation is limited, primarily due to the relatively low number of separate sequences and the static configuration within each sequence. Although learning from multiple frames of the same video is likely a useful thing to do in the context of deep neural networks[1], the frames within a video are correlated in that lighting conditions and relative object poses are the same.

Despite these limitations, we believe that the dataset is a proof of concept for a powerful idea: that strong generative models can be used to generate training data for discriminative learning-based techniques. While this dataset still required some human effort, it's easy to imagine a scenario in which the human can be phased out as the dataset grows. Xiang et al., (2018) were able to use the existing dataset (augmented with some synthetic data) to train a deep network capable of estimating object poses within the ICP basin of convergence over in over 90% of frames. The only reason humans are still needed in the creation of this dataset is that DART requires an initialization for object poses, but given this result, if more videos of these objects were collected one could use this trained network to give the initialization in most cases. One would expect this additional labeled training data to lead to better network performance. Ultimately, this could result in a 'virtuous cycle' in which more and more challenging sequences can be labeled as the network becomes more accurate, which is enabled by having more labeled sequences.

[1] Szegedy et al., (2013) identified a curious property of deep neural networks, which is that barely perceptible perturbations to network inputs can case drastic changes to network outputs. It is therefore likely useful to show a network many similar images (e.g. neighboring frames in a video) even if they appear nearly identical to the human eye.

In practice, a human was also used to record the data. However, there is no reason this couldn't be done with a robot-mounted camera. Furthermore, if the data were collected with a robot with end effectors (e.g. the SpaceJustin robot used in Chapter 3), the robot could manipulate the objects autonomously, actively collecting more varied data involving different object configurations.

We'll return to the idea of using generative models-based vision to aid in training deep networks in Chapters 6 and 7. First, now that we've explored the powerful applications of the DART tracking framework, we'll explore some of its limitations. These limitations will be used to further motivate the work in the chapters to follow.

Additional Resources

- The YCB-Video dataset can be downloaded here: https://rse-lab.cs.washington.edu/projects/posecnn/
- A video version of Figure 4.3 can be found here: https://youtu.be/V2ksiJkeS3Q

Chapter 5

The Data Association Problem

In previous chapters, we introduced the DART framework and demonstrated some of its powerful applications. In this chapter, we'll take a different view, looking at situations in which DART fails and what causes these failures. The next chapters will then explore ways to remedy the underlying issue.

5.1 Probing the Limits of DART

As a local gradient descent approach, DART is susceptible to falling into local minima. Its success is therefore entirely dependent on the size and shape of the basin of convergence, i.e. the region of parameter space from which a series of Gauss-Newton steps will lead to the correct solution. The basin of convergence is in turn heavily dependent on the model, the ground truth pose, and the observation, making it extremely difficult to analyze. However, we have performed some evaluations in order to give at least a baseline intuition into expected performance.

For these experiments, we use our human hand model and the point cloud depicted in the left panel of Figure 5.1. This combination of model and observation is interesting because the model is fully observable such that there is no ambiguity over the pose given global information, but, given the geometrically repetitive nature of the fingers, there are many local minima. This presents a challenge for a local gradient-based method attempting to recover the correct pose beginning from a distant location in pose space.

All experiments presented in this chapter hinge on measuring whether the tracker can recover the true pose from some initial starting point. Success of convergence is difficult to quantify in a binary sense, but for the purpose of these experiments, success means that the final estimated pose differs from the true pose by at most 5 mm in global translation, 2.5° in global rotation, and 10° in *any* of the joint angles. This means there are many instances in which the tracker mostly recovers the true pose but, for example, estimates the tip of one finger to be flexed when it is in fact extended. This error would cause the sample to be considered a failure.

Susceptibility to Global Position and Orientation Displacement

First, we evaluated the effect of displacement of the camera-to-root-frame transform on the basin of convergence, in order to give a sense of the ability to track fast, global motion of the model relative to the camera. This was done by sampling a large number of starting poses with various translational and rotational offsets relative to the ground truth pose. Articulations were not perturbed in this experiment. We show in Figure

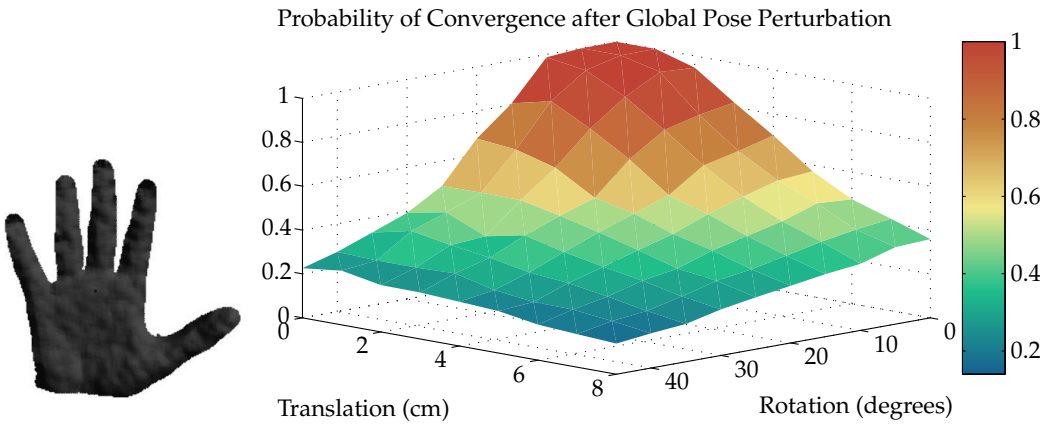


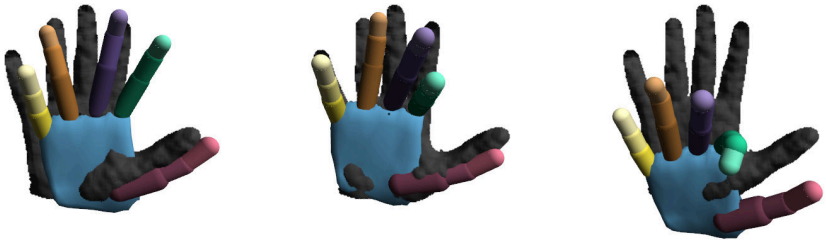
FIGURE 5.1: (left) The point cloud used for the basin of convergence experiments, rendered with estimated point normals. The point cloud was observed by a short-range time-of-flight sensor. (right) Basin of convergence in six degrees of freedom. The camera-to-model-root transform for the human hand model is displaced from the ground truth value for the observed point cloud shown on the left. The probability of convergence is estimated to be the percentage of samples which converge to the ground truth pose within 20 iterations.

5.1 the percentage of samples at each displacement that successfully recovered the true pose within 20 iterations, which is approximately the maximum number of iterations that can be performed for each frame in order to achieve real-time performance. In interpreting these results, note that at this frame rate, a 12 degree frame-to-frame rotation corresponds to a full revolution per second and likewise a 3.3 cm frame-to-frame translation corresponds to a linear hand velocity of one meter per second. As can be seen, the basin of convergence is stable within around 2 cm and 10° deviation from the true pose.

Susceptibility to Full Articulated Displacement

Next, we evaluated what happens when all joints of the articulated pose are displaced, in addition to the global rotation and translation of the root frame. In this case, the 'distance' from the true pose to a sample involves a norm in $\mathbb{R}^{26}$, a pose space which includes both translations and rotations and as such has no real physically-interpretable units. Figure 5.2 illustrates some displacements of various magnitudes to aid in the interpretation of the results.

Figure 5.3 shows the probability of convergence from the displaced starting points, both with and without the negative information term described in section 2.2.4. We also show here the effect of the number of iterations on the basin of convergence. For this particular pose, it is interesting to note that for small displacements, performing more iterations actually causes less samples to recover the 'correct' pose; this is because there is a local minimum very near the true minimum, in which the tip of the index finger is bent such that the observed points on the fingertip lie on the back of the predicted surface, rather than the front. There are then parts of pose space that, under our metric, are close enough to be considered correct, but have a gradient pointing away from the true minimum; thus, the tracker converges very slowly towards this local minima and

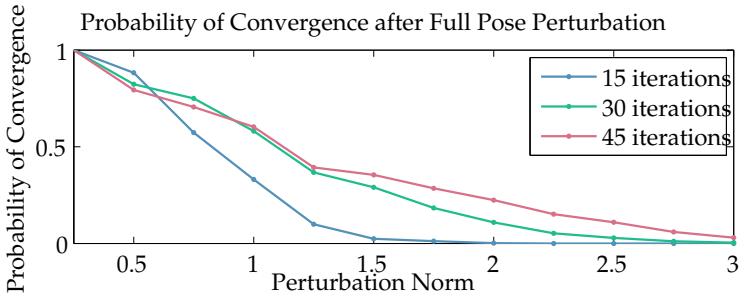


(A) norm=1, convergence successful (B) norm=2, convergence successful (C) norm=3, convergence successful

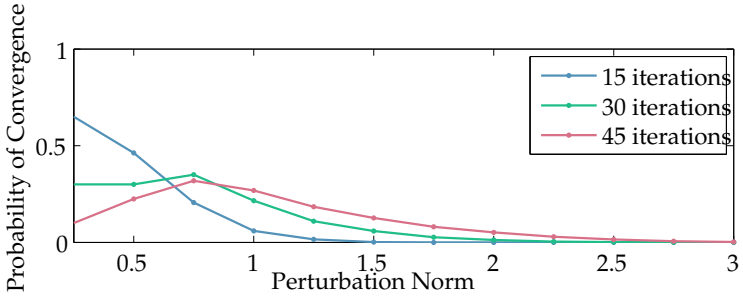
(D) norm=1, convergence unsuccessful (E) norm=2, convergence unsuccessful (F) norm=3, convergence unsuccessful

(G) norm=1, optimization result (H) norm=2, optimization result (I) norm=3, optimization result

FIGURE 5.2: Example results of convergence testing with perturbation of the full articulated pose. (A) and (D) show samples displaced from the ground truth pose by a norm-1 vector, (B) and (E) by a norm-2 vector, and (C) and (F) by a norm-3 vector. (A)-(C) converged successfully to the correct pose within 20 iterations. (D)-(F) either did not converge within 20 iterations or converged to a local minima. (G)-(I) show the pose estimates for samples (D)-(F) after 20 iterations. (G) and (H) show failure modes in which enough points are associated with the wrong finger to lead to a local minima. (H) also shows another failure mode for the last joint on the thumb; here, the joint angle is estimated to be fully flexed when it is fully extended, but according to the linearization around the current estimate it appears to the optimizer as if decreasing the joint angle will not effect the error. (I) shows a more dramatic failure mode in which a large error in the estimate of the global rotation of the hand results in a near total data association failure.



(A) With negative information (DART symm.).



(B) Without negative information (DART asym.).

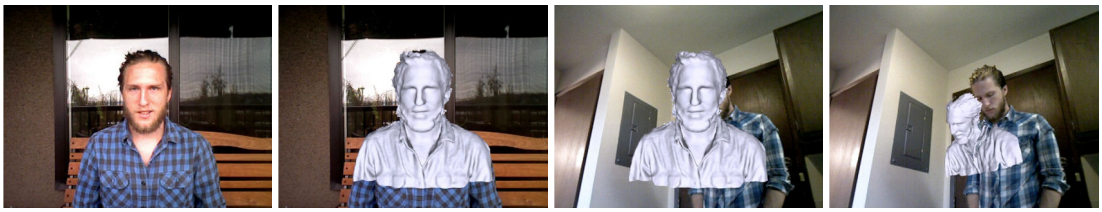
FIGURE 5.3: The experimental probability that the tracker will recover the correct pose starting from a pose at various distances from said correct pose, with and without the negative information term described in section 2.2.4. The distance is measured by a 'perturbation norm' that balances global displacement and displacement of the articulated joint angles.

away from the true solution. This effect is much more evident in Figure 5.3, as the use of free space information helps prevent convergence to this particular local minima.

Figure 5.2 shows some example results for samples from this evaluation. The displaced poses shown in (A), (B), and (C) all led to successful convergence to the true pose within 20 iterations. The displaced poses shown in (D), (E), and (F) also converged within 20 iterations, but to a specific local minimum, show in (G), (H), and (F). There are also some samples which simply had not converged within 15, 30, or even 45 iterations, and would have converged to the correct solution eventually. This visualization of failure modes provides some insight into what causes failure in DART. Only one joint angle in one of these examples (the thumb in (H)) cannot be estimated due to the non-linearity of rotation. All other failures shown are caused by the naïve data association technique in which every point is simply associated with the nearest point. When enough associations are estimated incorrectly, this leads to a failure to converge to the correct solution.

5.2 Data Association and SLAM Re-initialization

DART is not alone in facing the Data Association problem. On the contrary, data association (also known as correspondence estimation) crops up in nearly every computer vision problem. Dense reconstruction techniques such as KinectFusion (Newcombe et al., 2011) must solve a frame-to-model correspondence problem for each new observed



(A) (B) (C) (D)

FIGURE 5.4: An illustration of the challenges of relocalization without correct data association. (A) A frame from an RGB-D video of a person. (B) A rendering of a DynamicFusion model of the person is shown over the frame. (C) The model is initialized in a new video. Because the pose is not known, it maintains the pose estimate from the source video. (D) Optimizing the DynamicFusion objective function starting from this frame results in a local minimum due to incorrect data association.

frame. Like DART, it can usually rely on a high frame rate such that a naïve data association technique (in this case, projective data association) yields an initialization for each new frame which is within the basin of convergence for gradient descent.

Dense reconstruction systems also face the data association problem when closing loops, i.e. when a part of the reconstructed model leaves the field of view and returns at a later time. If there has been drift in the state estimates, it cannot be assumed that a simple data association technique (such as nearest neighbor in Euclidean or projective space) will lead to convergence, i.e. a closed loop. Prior work has investigated solutions to this problem, such as ElasticFusion (Whelan et al., 2016), which uses random ferns to recognize locations that it has seen before.

Often loop closure is aimed at closing loops when locations are revisited within a few minutes, as is the case with ElasticFusion. However, the difficulties of estimating correct data association are exacerbated when there are large appearance changes between observations of the same model, which often happens when revisitations are more separated in time. For example, in outdoor scenes, large temporal gaps between observations can lead to rather extreme illumination changes. Much of the research on this problem is focused on the autonomous driving application domain (c.f. (Milford and Wyeth, 2012; Maddern et al., 2017)). However, the task for autonomous driving is often the localization of the car according to a captured image, and while the car pose technically has six degrees of freedom it almost always lies in a three-dimensional manifold due to the fact that it is generally resting on the road surface. Furthermore, the goal is often to become robust to 'distractor' objects such as pedestrians and other vehicles. In the more general data association problem, we would like to be able to estimate correspondence for dynamic objects as well.

Figure 5.4 gives a concrete example of the difficulties of data association. In this case, a model of a person is reconstructed using DynamicFusion (Newcombe, Fox, and Seitz, 2015) from one RGB-D sequence, and the goal is to re-initialize the model state in a new sequence in order to continue tracking the model. No information is known about the pose of the model in the new sequence, so the pose is initialized with the canonical pose from the original sequence. Unfortunately, this does not lead to convergence to the correct pose when optimizing the DynamicFusion objective function. One option would be to use random restarts, run the optimization, and keep the converged pose

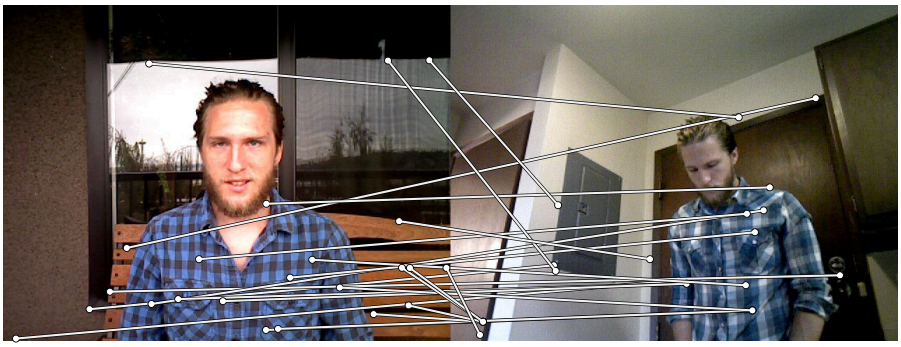


FIGURE 5.5: Under strong viewpoint, illumination, and non-rigid pose changes, engineered local descriptors also fail to identify the correspondences needed to recover the pose of the model shown in Figure 5.4. Here we show SIFT matches for the same image pair which pass the standard ratio test (Lowe, 2004).

with the highest likelihood. However, models such as the articulated objects tracked in Chapter 2 and fully-nonrigid models have high-dimensional state spaces which make searching in pose space extremely expensive. Even further complicating matters is the fact that data association also plays a role in the likelihood function of systems such as DART and DynamicFusion, such that an incorrect state estimate may appear likely with incorrect data association but is in fact not the true state. Detecting such a situation would also require knowledge of the ground truth data association.

If enough correct correspondences were known these problems could be solved. Unfortunately, many traditional correspondence estimation techniques such as local feature matching are insufficiently robust to viewpoint, illumination, and pose changes, and are not typically useful in the absence of texture. The result of SIFT matching for the two images from Figure 5.4 is shown in Figure 5.5, with no meaningful correct matches found.

Discussion

In this chapter, we looked at some of the ways in which model-based tracking alone can fail. One of the weakest points in many model-based tracking pipelines is a naïve estimation of data association which works in many cases, especially in frame-to-frame tracking scenarios. However, to enable truly robust tracking (and reconstruction) and enable efficient relocalization of known models, we will need better data association techniques.

In Chapter 4 we explored an idea in which model-based tracking can be used to supervise deep networks designed to estimate object poses. In the next chapter, we'll push this idea further by showing how dense reconstruction techniques can be used to supervise deep networks designed to produce dense descriptors, even when no models are available a priori and even when the model moves non-rigidly. We'll also explore how these dense descriptors can be used to help solve challenging data association problems by describing objects at a more abstract, semantic level that is more invariant to viewpoint, illumination, and pose, and allows for reasoning about correspondence even for textureless surfaces.

Chapter 6

Self-Supervised Dense Descriptor Learning

Robots operating in dynamic environments with humans present will depend on solutions to a plethora of perception problems, such as simultaneous localization and mapping (SLAM) with loop closure detection, object recognition and tracking, and human body tracking. As discussed in the previous chapter, solutions to these problems depend in turn on solving the correspondence problem in one form or another, allowing robots to relate their live visual observations to previous observations or to a known model. Sparse features such as SIFT (Lowe, 2004) can be helpful for localization of a camera or tracking of an object in six degrees of freedom, so long as there is sufficient texture. However, when localizing in a poorly textured scene or tracking a possibly deforming model with many degrees of freedom (e.g. a mesh model of a person), dense correspondences can be crucial for resolving ambiguities or locking down degrees of freedom. While manually-designed heuristics, such as computing histograms of local gradients, lead to strong sparse descriptors, it is not at all obvious how one might engineer a powerful dense descriptor, especially given the paucity of information in small, local patches located far from highly discriminative features. We therefore are interested in learning dense descriptors that take a large amount of context into account.

Recent trends in machine learning have shown that deep neural networks can be used to learn strong features for various tasks given a sufficient amount of training data. Work on deep networks in the machine learning and computer vision communities tends to focus on learning models with strong generalization performance within classes of objects, such that from a number of labeled images of a particular class the network can learn to recognize when novel images depict new instances of the class (Krizhevsky, Sutskever, and Hinton, 2012). In this work, we will take an approach informed by the embodied vision paradigm, focusing instead on generalization performance at an instance level. In other words, we want to enable a robot to learn to recognize novel views of an object or environment that it has seen before. This level of generalization can be extremely useful for many robotics applications, such as initializing a model-based human body tracking system or detecting and closing loops in a known environment. Furthermore, as long as it is inexpensive to acquire training data, the set of visual concepts the robot knows about can be easily expanded.

In order to obtain training data at little cost, we return to the idea put forth in Chapter 4 and use a generative model-based technique to automate the labeling process. However, instead of relying on the DART tracking framework which requires that the geometry is known a priori and must be initialized (typically manually), in this chapter

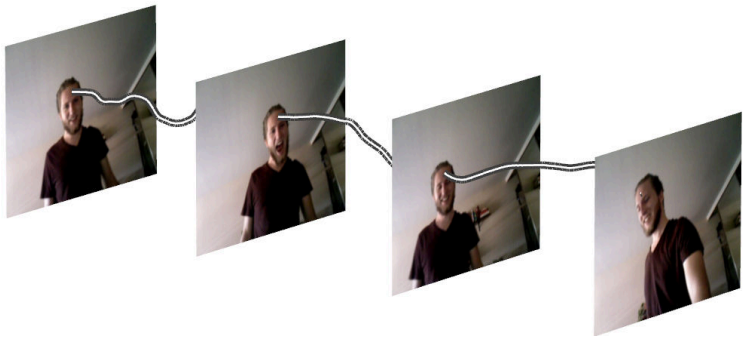


FIGURE 6.1: A visualization of the data provided by DynamicFusion. The line shows the trajectory of a point on the densely tracked model, with the left-to-right dimension representing time. Note that the trajectory crosses the same part of the forehead in all example frames shown.

we avoid both problems by using state-of-the-art dense SLAM methods that build and track models through RGB-D videos (c.f. Figure 6.1). Specifically, we use KinectFusion (Newcombe et al., 2011), for rigid scenes, and DynamicFusion (Newcombe, Fox, and Seitz, 2015), for non-rigidly deforming scenes. These techniques rely only on very basic assumptions and require no templates. They track the state of a reconstructed mesh model and can therefore be used to identify a set of dense correspondences within a video without any human intervention.

While one could train a neural network to produce dense coordinate maps of the automatically-constructed models by regressing the reprojected vertex coordinates, this leads to a problem if there are multiple videos of the same object or environment. In this case, each map is in a coordinate system that is local to the video in which it is constructed. Registering these local models into a global coordinate frame would require a solution to the correspondence problem, which is what we are trying to solve. While alignment of rigid scenes in six degrees of freedom is fairly straightforward, it is much more complicated when objects move between visitations, and alignment of non-rigidly deforming models is an even more challenging problem. We are therefore interested in learning methods that operate on a local labeling which can be produced automatically.

In order to use only local labelings to learn a descriptor that can be used as a global similarity measure, we can only make use of relative labels (i.e. labels identifying relationships between pixel observations rather than absolute properties of single pixels). We therefore adopt the contrastive loss of Hadsell, Chopra, and LeCun, (2006) and Chopra, Hadsell, and LeCun, (2005), which uses a label for every pair of inputs identifying whether the inputs are from the same class or not to learn a latent representation with the property that instances of the same class are clustered and separated from instances of other classes. We extend this idea to dense and continuous labelings, and consider pixels with the same local model coordinates as tracked by a generative model to be of the same 'class,' and consider all other image pairs from the same video to be of different 'classes.' By using only relative labelings, we can learn from multiple videos of the same instance simultaneously without needing to align the models. The only question that remains is whether the network will learn a model that is consistent across the videos, such that correspondences between the models can be subsequently identified

FIGURE 6.2: Example frames from both datasets. Note the viewpoint, lighting and pose variation.

using the learned descriptor, or whether the network will learn a separate descriptor space for each video, as both are valid solutions under the contrastive loss when image pairs are only selected from within the same video.

We show, in a somewhat surprising result, that if a network is trained using a contrastive loss on many automatically-labeled videos of the same instance (object or environment), the resulting descriptor is to a large degree consistent across videos and can be used to identify dense correspondences in novel images. Apparently, the variability in viewpoint, lighting and deformation (in the case of the human data) in each local video sequence is enough to teach a network to be invariant to these factors. We have tested our approach on two datasets we collected ourselves which focus on revisitations of the same model, in one case a human head and torso and in the other an office. These are challenging datasets that include large variations in clothing and background for the former, and different lighting conditions and arrangements of objects for the latter (see example frames in Figure 6.2). We also test on a publicly available set of videos of seven different strictly rigid scenes and show that we can train a single network that can be used for state-of-the-art camera localization in held-out test videos of the same seven scenes. We believe our approach is highly relevant in the robotics domain as it allows robots to gather their own training data and learn to better understand their local environment without requiring any human input.

6.1 Visual Descriptors and Deep Metric Learning Overview

There exists a large body of work on hand-engineered image descriptors (Bay, Tuytelaars, and Van Gool, 2006; Lowe, 2004). For brevity, we here focus on learning descriptors, which has received significant attention in the computer vision community.

Much of the related work on learning for dense correspondence estimation has taken advantage of absolute labels. For example, Taylor et al., (2012), Shotton et al., (2013), and Brachmann et al., (2014) train random decision forests to assign a corresponding coordinate on a known model to every pixel in an image. Semantic segmentation networks,

such as those by Long, Shelhamer, and Darrell, (2015) and Hariharan et al., (2015), produce dense classifications of image pixels, but also require dense class labels for training. Because of the absolute nature of the labels used by all of these systems, they tend to rely on manual labeling for training, which severely limits the size of dataset that can be collected at reasonable expense. We, on the other hand, are able to generate our own labeled data at almost no cost.

Other prior work has investigated the use of relative labels amongst data points. Chopra, Hadsell, and LeCun, (2005) use a contrastive loss to train a convolutional neural network to recognize whether two images of human faces are images of the same person or not. More recently, Wohlhart and Lepetit, (2015) use pairwise labels to learn an embedding that encodes the relative pose of a known object, and show that the resulting embedding can be used for relative object pose estimation. However, their approach is at an object level, requiring the object to be centered in the image, and does not produce dense representations. Wang and Gupta, (2015) track patches through YouTube videos and use a triplet loss to train a network to produce feature representations of two patches from the same track that are nearer to each other than they are to a third patch not on the track. They also demonstrate the promise of relative labels, but the granularity of their tracking approach is not nearly as fine as one can achieve with a strong geometric model-based tracking system. Furthermore, because they have such a vast source of training data, they do not focus on re-visitation, so we do not know if their learned features would be able to identify re-visitations in the training data. Approaches similar to ours have been used (e.g. by Zbontar and LeCun, (2016), Choy et al., (2016), and Gadot and Wolf, (2016)) to identify dense correspondence in images for the purpose of estimating optical flow. Another interesting recent technique for optical flow that is competitive with deep learning approaches but does not require any learning is DeepMatching (Revaud et al., 2016). However, optical flow methods tend to focus on much shorter-range correspondences than we are capable of identifying with our learned descriptors, and in general we are interested in frame-to-model correspondences in addition to the frame-to-frame correspondences that are the domain of optical flow.

The use of model-based reconstructions to build datasets for learning to do correspondence estimation goes back at least to the work of Winder and Brown, (2007), in which a multiview stereo reconstruction of a few landmark locations was used to identify the ground truth correspondences. More recent work, such as that of Simonyan, Vedaldi, and Zisserman, (2012), Simo-Serra et al., (2015), Zagoruyko and Komodakis, (2015), and Han et al., (2015), also make use of relative labels as given by a generative model, in this case Phototourism reconstructions of landmarks (Snavely, Seitz, and Szeliski, 2006). While fine-grained, the images in these datasets are already known to be discriminative given their use as keypoints in Phototourism, and furthermore the full set of relations between patches is given, which means that applying these methods requires global alignment of all known patches. Tompson et al., (2014) track a human hand using a generative model, and use the resulting poses to train a CNN to detect 2D locations of keypoints on the hand. This approach also requires global alignment, in this case to a known kinematic hand model. Zeng et al., (2017) also makes use of a generative model based on dense fusion to provide automated labeling, computing descriptors on 3D fused geometry fragments at sparse 3D keypoints. They also consider multiple visitations to the same environment and also test on the same seven scenes, but

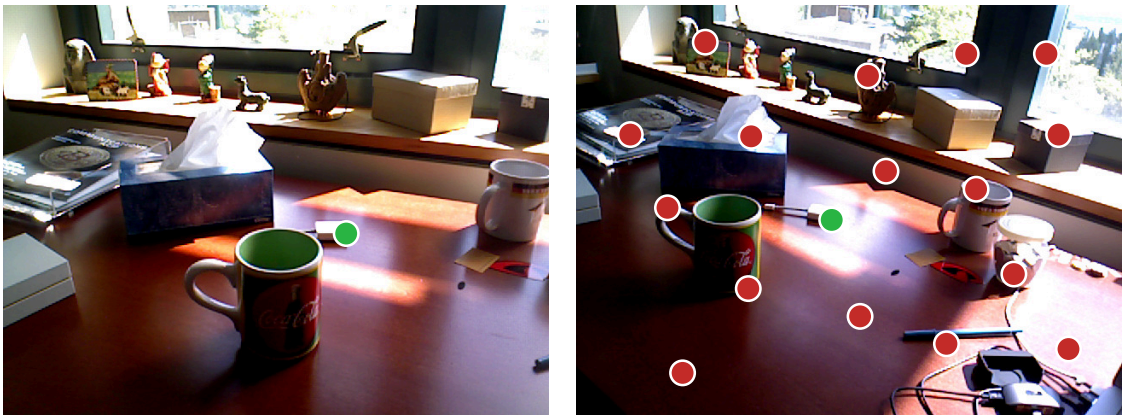


FIGURE 6.3: An example image pair as used for training our networks. For each sampled pixel location from the left image, we apply the contrastive loss at the corresponding pixel location in the right image (green) and at randomly selected non-corresponding locations (red).

they make use of the known alignment between videos of the dataset to label all pairs, whereas we use only labelings within a single video.

Existing SLAM systems that utilize feature descriptors have traditionally used hand-engineered features, such as ORB-SLAM (Mur-Artal and Tardós, 2017), FabMap (Cummins and Newman, 2008), and BundleFusion (Dai et al., 2017). A notable exception is ElasticFusion (Whelan et al., 2016), which uses random ferns. These approaches demonstrate how robust correspondence detection can be used to great effect in SLAM systems and are orthogonal to the work presented in this chapter, as our feature learning approach could be used to estimate correspondences instead. Sünderhauf et al., (2015) use AlexNet features pre-trained on ImageNet for visual place recognition. As we will show, we are able to learn features that are more robust and discriminative than AlexNet features on known objects and environments that have been seen before. Lowry et al., (2016) provide a useful overview of visual place recognition, the challenges involved, and the importance of identifying correspondences for robotics applications.

6.2 Training Method

To formalize the problem, we are interested in learning a non-linear function $f(x) : \mathbb{R}^{W \times H \times 3} \rightarrow \mathbb{R}^{W \times H \times D}$ which maps $W \times H$ color images to a dense descriptor, representing each pixel with a vector of length D. We would like the descriptor to encode the identity of the point that projects onto the pixel. Ideally, the representation would be invariant to lighting, viewpoint, deformation, and all other variables other than the identity of surface that generated the observation. If we are able to successfully learn such a descriptor, we can use a simple distance metric such as Euclidean distance to identify dense correspondences between pixels in two images.

6.2.1 Convolutional Neural Network Architecture

We use a convolutional neural network to learn our non-linear function. In order to efficiently compute dense representations, we adopt the Fully-Convolutional Network architecture of Long, Shelhamer, and Darrell, (2015), which yields strong results for the related task of dense semantic segmentation. This architecture uses deconvolutional layers applied to spatially-pooled feature maps in order to produce an output feature map with the same spatial resolution as the input. The only modification we make to the FCN is to the output dimensionality.

6.2.2 Training

Our approach is designed for a specific training regime, formalized as follows. Assume we have a collection of V videos, where each video $v_i = \{(I_0, X_0), ..., (I_K, X_K)\}$ is a set of K frames, where for each frame k we have an image I_k and a corresponding dense model vertex map X_k which gives a mapping from image pixels in I_k to video-specific model coordinates. These coordinates are provided automatically by a SLAM system that constructs the model and tracks it throughout the video. The trajectory of a particular model vertex through a video is visualized in Figure 6.1. While we have many videos of the same instances, the coordinate system defined by the vertex maps $X_k \in v_i$ are not consistent with the coordinate systems given by vertex maps $X'_k \in v'_i$ for $v_i \neq v'_i$. For example, given a video of a person, DynamicFusion can map all appearances of the tip of the person's nose to a particular model vertex, but in a different video of the same person the tip of the nose will be assigned a different coordinate depending on their initial pose.

Because we only have model coordinates in local frames of reference, we cannot regress to a single canonical surface representation as did Taylor et al., (2012) or learn to recognize a set of canonical keypoints following Tompson et al., (2014). We thus take the approach of Hadsell, Chopra, and LeCun, (2006) and use pairwise labelings of images, which in our case indicate whether or not two pixels are projections of the same model point from the same video. Given our descriptor function $f(x)$ parameterized as a convolutional neural network, we define a pixel-level contrastive loss as

$$L(I_a, I_b, \mathbf{u_a}, \mathbf{u_b}, X_a, X_b) = \begin{cases} d(I_a, I_b, \mathbf{u_a}, \mathbf{u_b})^2 & X_a(\mathbf{u_a}) = X_b(\mathbf{u_b}) \\ \max(0, M - d(I_a, I_b, \mathbf{u_a}, \mathbf{u_b}))^2 & \text{otherwise} \end{cases}, \quad (6.1)$$

where M is a margin (set to 0.5 in our experiments) and $d(I_a, I_b, \mathbf{u_a}, \mathbf{u_b})$ is a distance metric between the descriptor of image I_a at pixel $\mathbf{u_a}$ and the descriptor of image I_b at pixel $\mathbf{u_b}$ as defined by

$$d(I_a, I_b, \mathbf{u_a}, \mathbf{u_b}) = ||f(I_a)(\mathbf{u_a}) - f(I_b)(\mathbf{u_b})||_2 \,, \quad (6.2)$$

where $f(I)(\mathbf{u})$ denotes using a pixel value $\mathbf{u} \in \mathbb{R}^2$ to compute a descriptor vector in $\mathbb{R}^D$ via bilinear interpolation of the dense descriptor $f(I)$.

In practice, we compute dense descriptors for images at VGA resolution, such that each image has hundreds of thousands of descriptors. Thus, considering all the pairwise matchings between two images would lead to tens of billions of evaluations of

the contrastive loss function. Instead of considering all possible pairs, we sample locations at which to apply the contrastive loss. We first choose up to 5,000 pixel locations $\mathbf{u_a}$ uniformly at random from the set of pixels that have a correspondence in I_b. For each sample from I_a, we apply the loss function to the pair formed by $\mathbf{u_a}$ and the corresponding pixel from I_b (first case of equation 6.1), and then to 100 pairs formed by $\mathbf{u_a}$ and non-corresponding pixel locations chosen uniformly at random (see Figure 6.3). The loss for all positive pairs is divided by the number of matches, and likewise for all negative pairs such that the loss is not biased by the larger number of negatives.

Our networks were trained from scratch using the caffe software framework (Jia et al., 2014), with a modified contrastive loss layer that takes as input two images, two vertex maps, and samples point pairs, computing the loss defined in equation 6.1. During backpropagation, the loss of each sampled pair is added to the appropriate location in a dense gradient map so that the losses of all samples propagate backwards through the network simultaneously. Unless otherwise specified, our networks are trained with $D = 32$.

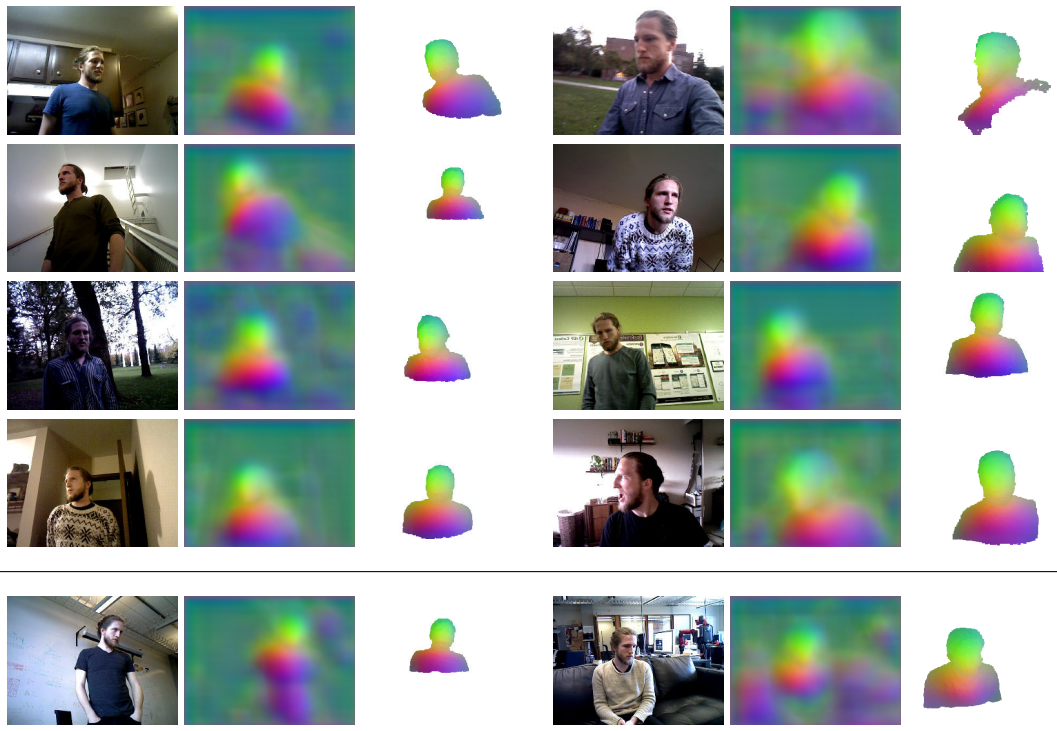


FIGURE 6.4: A visualization of our features trained on the human dataset with output in $\mathbb{R}^3$. In both of the repeated sets of three columns, the left column shows frames from various sequences, the middle column shows the normalized output of the dense correspondence network for these frames, and the right column shows descriptor values from all frames in the sequence fused onto the DynamicFusion model. Images above the line are from sequences used in training, and images below the line are from held-out test sequences. Note that the network uses a very similar mapping for all rows despite having no cross-sequence training signal, and that the mapping generalizes well to novel sequences.

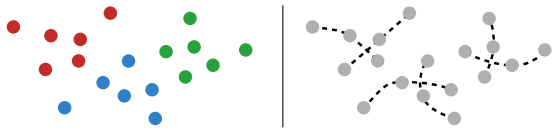


FIGURE 6.5: An illustration of our problem formulation in two dimensions. At left is the traditional classification task in which a number of observations are given along with class labels (encoded by color). The goal is then to learn the boundary between classes. At right there are no labels, but there are connections indicating shared identity between observations. Given a dense enough sampling of observations, the separation should still be learnable.

6.3 Results

To illustrate the outcome, we first present qualitative results on a dataset of videos of a person modeled in different settings and lighting conditions. We then present quantitative comparisons between our learned descriptors and existing descriptors on a benchmark dataset and on our novel datasets generated with KinectFusion and DynamicFusion. Finally, we demonstrate that our learned descriptors can be used to initialize a deformable mesh model of a person wearing different clothes and under drastic changes in lighting conditions.

6.3.1 Qualitative Result

For qualitative evaluation purposes, we trained a network with $D = 3$ on the human dataset, which consists of 62 videos of the same person in different locations and wearing different clothing. Because the network output is in $\mathbb{R}^3$, it can be visualized by simply mapping the descriptors directly to the RGB color cube. In Figure 6.4, we show frames from the videos, network output for these frames, and the average network output fused onto the model surface.

Recall that we only apply the loss function to pairs of images for which the correspondences are known, which requires that both images in the pair come from the same video. Therefore, the network was never given any information regarding the relationship between pixels in separate videos. Somewhat remarkably, the network has learned a descriptor which, for the most part, maps points consistently across videos. This result is entirely data-driven – a similar mapping in which the network produced different mappings for each video would have had equal loss under our formulation in section 6.2.2. However, the network has apparently learned that it can most easily learn to simultaneously minimize the loss function for patches selected from each individual video by using a shared representation.

We illustrate why we think this works in Figure 6.5. Essentially, we rely on the fact that patches sampled from a model point trajectory will inevitably come near in the ambient space to a patch from another trajectory that tracked the same model point in another video. By chaining together a number of such convergences, the underlying structure starts to emerge even without a full labeling. This requires a large number of trajectories, but our results seem to indicate that it does not require an unreasonably large number, especially if the data can be collected autonomously.

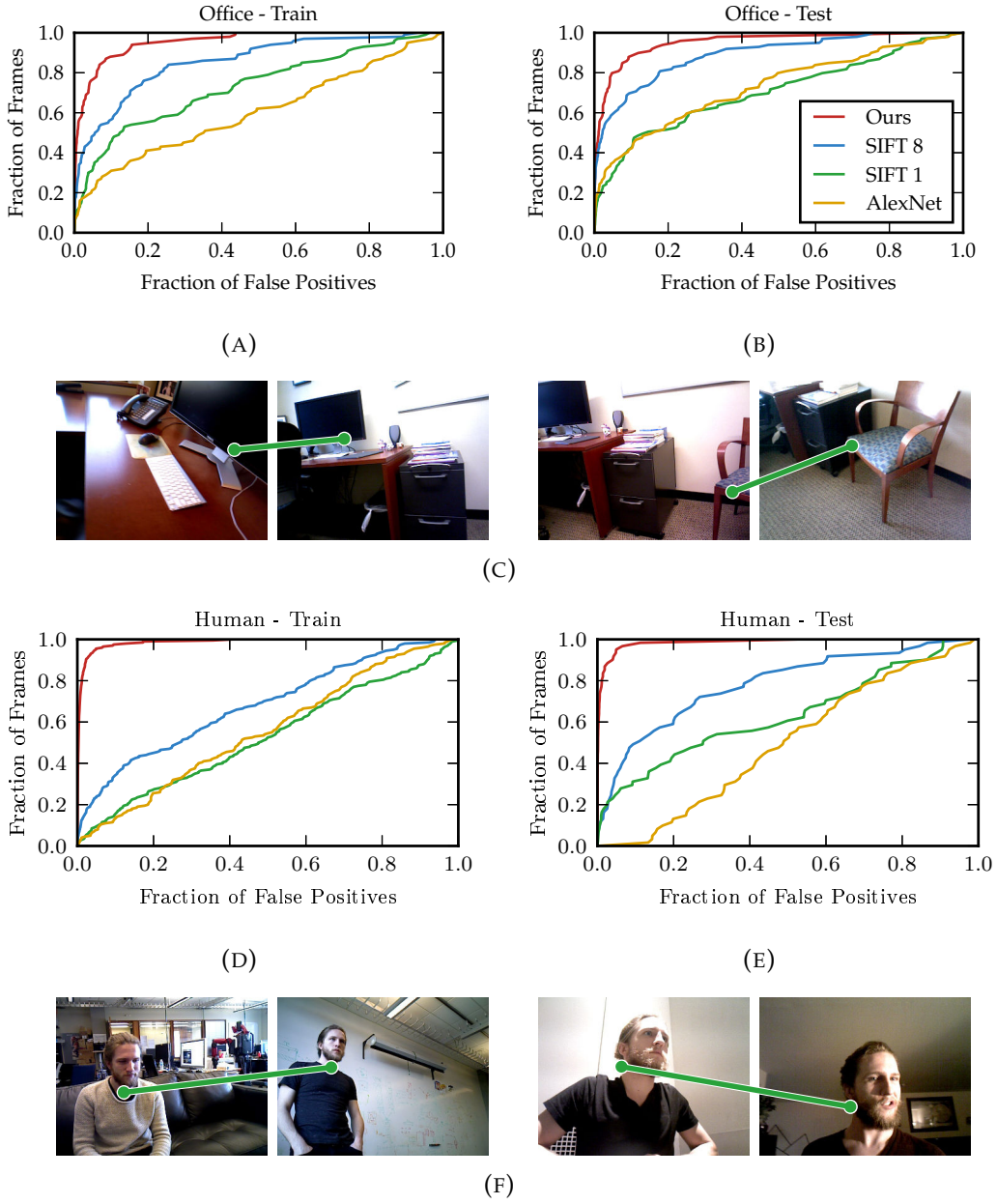


FIGURE 6.6: Quantitative results for both datasets. (A) and (B) show frame-to-frame point matching results for the office dataset, and (C) and (D) show results for the human dataset. (A) and (C) show results for matching points across training videos, and (B) and (D) show results for matching points across held-out videos. (C) shows the ground truth match for some test cases from the office dataset, and (F) shows the same for the human dataset. Note the extremely challenging variations across videos. The number associated with the SIFT results indicates the scale at which the SIFT features were evaluated. While the larger scale gave better performance, it also resulted in much slower evaluation (for reference, the FCN network computes a dense 640 x 480 descriptor in about 50 ms on a modern GPU).

6.3.2 Quantitative Results

We also performed a quantitative evaluation of our networks, analyzing the point correspondence accuracy in frame-to-frame settings and the relocalization accuracy for a frame-to-model setting.

SIFT Baseline

For this result, we compare our learned features against a dense SIFT representation on the task of finding the point in a target image that corresponds to a given point in a source image. This is done for both our human and office datasets. While we have labels within each video of our datasets, we do not have any labels spanning video boundaries. To evaluate our descriptors, we therefore manually labeled pairs of corresponding points from different videos. We consider two settings, one in which the labeled pairs come from the training videos, such that the network has seen the images before but has never been told what the correspondences are, and one in which the pairs come from held-out videos that the network had never seen before in any context.

Given the labeled pairs, we extract the descriptor value for the source point in the source image, and compute a dense descriptor for the target image. We then determine the number of pixels in the target image that are closer in descriptor space to the source point than the manually-labeled corresponding point. The results are shown in Figure 6.6. On both datasets, using our descriptor allows us to identify more correspondences with less false positives than when using a dense SIFT descriptor. We also compare against a dense descriptor formed by computing AlexNet 'fc6' features, showing that by training on domain-specific data with a fine-grained loss function, we far outperform a feature descriptor trained on ImageNet with a coarse loss function.

MSR 7-Scenes

The MSR 7-Scenes dataset by Shotton et al., (2013) consists of multiple RGB-D videos of seven different rigid scenes, divided into a standard training and test set, with at least one training video and one testing video per scene. The number of videos per scene varies between 2 and 12. There is also a TSDF model for each scene and a camera pose for every frame of every video which is in the global frame of reference of the TSDF model.

Valentin et al., (2015) showed state-of-the-art results on the task of estimating the camera pose for all frames in the held-out test videos, using a random forest that was trained to regress the global model vertex coordinate that projects on each pixel. Their random forest maintained at each leaf node a probability density function in $\mathbb{R}^3$ encoding the uncertainty over model coordinates. They evaluate the forest at sparse locations selected randomly from the image and optimize the pose using RANSAC.

For comparison, we first train a network using the procedure described in section 6.2, where we ignore the alignment between videos that is given implicitly by the fact that the camera poses are in a global frame. In other words, we only sample pairs from the same video such that if the camera poses in each video underwent an affine transformation, our results would be unaffected (the same is not true for Valentin et al., (2015)). We then compute a dense descriptor for every frame of the training videos and compute the mean descriptor values onto which a densely sampled set of points on the

TABLE 6.1: Percent of MSR 7-Scenes Test Frames Localized within 5 cm and 5 degrees of Ground Truth

Scene	Sparse RGB (Shotton et al., 2013)	Valentin et al., (2015)	Our Method
Chess	70.7%	**99.4%**	97.75%
Fire	49.9%	94.6%	**96.55%**
Heads	67.6%	95.9%	**99.8%**
Office	36.6%	97.0%	**97.2%**
Pumpkin	21.3%	**85.1%**	81.4%
Kitchen	29.8%	89.3%	**93.4%**
Stairs	9.2%	63.4%	**77.7%**
Average	40.7%	89.5%	**92.0%**

model surface project (analogous to the right column of Figure 6.4). Finally, to estimate poses in the model frame of reference, we adopt the same RANSAC algorithm as used by Shotton et al., (2013), with a few modifications. In order to reason about uncertainty over model vertex correspondences, as Valentin et al., (2015) did, we compute for each pixel sample the set of all model vertices with a mean descriptor value within a fixed distance of the pixel descriptor. Then, when deciding which pose samples to preempt, we compute the observed 3D point in camera coordinates at each pixel sample by back-projecting the depth, transforming the point into the model frame of reference using each pose sample, and consider for that pose sample only the closest model point out of the set of possible correspondences. This closest point is also used in the pose refinement step. Because we have such rich information from each pixel sample, we are able to use only 100 pixel samples per RANSAC iteration. We use 1024 initial pose samples following Shotton et al., (2013), and a slightly smaller inlier threshold of 5 cm. For more details on the RANSAC procedure, see Algorithm 1 in Shotton et al., (2013).

The results can be seen in Table 6.1. Despite the fact that our network was only given labels describing relationships between points in images from the same video, it learned a descriptor that was consistent enough to localize against novel views of the same scene at a level of performance that is near or beyond the previous state-of-the-art, which was trained to regress model vertex coordinates directly. Furthermore, Valentin et al., (2015) train a separate model for each of the seven scenes, whereas we use the same network for all scenes. While we did have to use the registered model to fuse descriptors across the training sequences, this was only so as to get adequate coverage of the model to be able to do global localization. We could have computed mean descriptors for each video separately and then localized against the partial, local models as we will do with the human models in section 6.3.3.

A descriptor in $\mathbb{R}^{32}$ is more difficult to visualize then a descriptor in $\mathbb{R}^3$, but we make an attempt in Figure 6.7 with a parallel coordinate chart, plotting each point as a line crossing 32 parallel coordinate axes, each representing one dimension of the descriptor. We arbitrarily chose three points from the 'office' model and plot all of their descriptors on all sequences for this scene (including test sequences). The descriptors for each point are clustered, but also well separated from each other (note that linear separability on one axis is a sufficient, but not necessary, condition for linear separability in 32 dimensions). We purposely chose points that do not coincide with strong visual features, showing our model's ability to use context to disambiguate points in

texture-poor image regions.

6.3.3 Application to Deformable Model Relocalization

Our approach can also be used to train networks that are useful for relocalizing the pose of deformable models. Recall that the reason we cannot use inter-sequence training pairs is that we have no way to automatically align models constructed from separate videos. We use our trained descriptors to provide just such a mapping by extending the RANSAC approach of Shotton et al., (2013) to non-rigid pose estimation.

The nonrigid poses we estimate are parameterized using the deformation graph described by Newcombe, Fox, and Seitz, (2015), which consists of a hierarchy of nodes which each have a canonical location and a six degree of freedom transform from the canonical frame to the observed target frame. These nodes then define a continuous deformation field via interpolation which can be used to warp the mesh model.

Whereas mean descriptor values in the previous section were computed at randomly sampled model vertices, we now compute mean descriptor values for each deformation graph node at the base level of the hierarchy. We again start with 1024 samples initialized by choosing three image points at random, computing the closest model points, and initializing the rigid pose of the model using Procrustes analysis. Because there are also background pixels in this setting, we immediately preempt any initial pose with a mean point-to-point error above 20 cm as these samples are highly likely to result from the selection of at least one background point.

We then continue as in Algorithm 1 of Shotton et al., (2013), but with the rigid pose refinement in line 12 replaced by a non-linear pose refinement. This is done via two iterations of the Gauss-Newton algorithm to minimize

$$E(\theta) = \lambda E_{ARAP}(\theta) + \frac{1}{|I|}\sum_{i \in I} ||x_i - m_i(\theta)||_2^2 \,, \tag{6.3}$$

where E_{ARAP} is the as-rigid-as-possible regularization term (c.f. Equation 8 in Newcombe, Fox, and Seitz, (2015)), I is the set of inlier indices for the pose being optimized, x_i is the i^{th} 3D observation, and $m_i(\theta)$ is the 3D location of the estimated corresponding model point m_i given pose θ. Finally, we minimized the full DynamicFusion objective function starting from the final nonrigid pose estimate given by RANSAC. Examples of

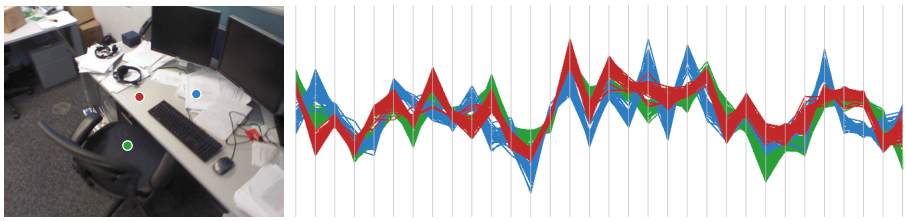
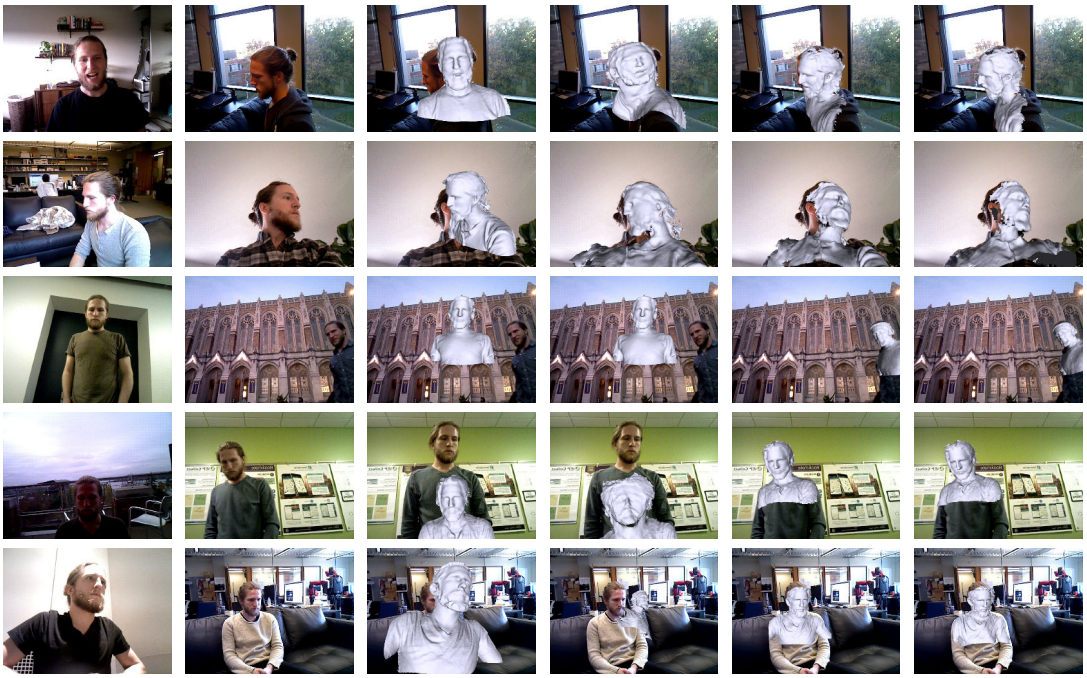


FIGURE 6.7: At left, three points in an image from the 'office' scene in the 7-scenes dataset are highlighted. At right, all of the 32-dimensional descriptors for these three points in all videos (including test videos) of the office scene is shown on a parallel coordinate chart, with the colors of the lines corresponding to colors of the points.



(A) (B) (C) (D) (E) (F)

FIGURE 6.8: Each row represents an attempt to localize a DynamicFusion model built from one video in another video, with and without our learned descriptor. (A) The first frame of the source video used to construct the 3D model. (B) A target frame from a separate sequence for which we would like to estimate the pose of the model. (C) The model built in the source video projected onto the target video, transferred via the identity transform. (D) The result of optimizing the DynamicFusion error starting from the identity transform pose shown in (C). (E) The initialization provided by running the RANSAC procedure described in section 6.3.3. (F) The result of optimizing the DynamicFusion error starting at the pose shown in (E). It is remarkable that alignment is possible using the learned descriptor despite the changes in clothing, lighting, and pose, and despite the fact the network was trained with no inter-sequence training signal. The first four rows are taken from videos used for training, but the last row shows a model built from a held-out test sequence being relocalized in another held-out test sequence, indicating a potential application of our system to relocalization from live input.

the result can be seen in Figure 6.8 and in the accompanying video. Because DynamicFusion is designed to track locally from one frame to the next, and because we have no idea where to expect to find the model in an entirely separate video, it is essentially coincidental if initializing with the identity transform happens to land within the basin of convergence. However, the RANSAC procedure requires no initialization, and our network has learned features that are fine-grained and discriminative enough to robustly estimate a nonrigid pose initialization that is within the basin of convergence of the local DynamicFusion objective. This helps solve the model transfer problem described in Chapter 5.

Discussion

This work introduced a paradigm that enables robots to learn visual features in a fully self-supervised way. The key idea is to leverage dense mapping techniques such as KinectFusion and DynamicFusion to provide labels indicating which pixels in a video sequence are projections of the same surface location in a model of an object, person, or environment. These within-video correspondences are used to train a CNN that computes dense visual descriptors that can be used to discriminate between different points on a model surface but are invariant to pose, viewpoint and lighting conditions. Importantly, we show that the learned descriptors are consistent across multiple videos of the same person or environment, enabling our framework to learn data associations across a wide range of conditions. We validated our approach by showing that the learned descriptors outperform a hand-engineered feature at estimating correspondence, and by showing that learning a descriptor using only relative labels on subsets of a publicly available dataset can outperform a technique that made use of absolute, spatially-aligned labels across the entire training set on a camera relocalization task. We furthermore show with the human dataset that our learned descriptor is robust enough to be used to accurately relocalize the high-dimensional pose of a deformable model. We did not need to hand-engineer our features, label any data manually, or do any manual template modeling.

We believe that utilizing robust generative models such as KinectFusion or DynamicFusion which require minimal training (or none at all) to generate training data is an extremely promising way forward for embodied vision. By taking advantage of the strong mapping techniques already at our disposal, we can automatically extract a training signal from raw video data. As we have shown, such automatically-labeled local structure is sufficient to train deep networks that produce descriptors for robust data association. The descriptors learned through our framework have immediate applications in various important problems in robotics, including simultaneous localization and mapping with loop closure detection, object recognition and tracking, and human body tracking. An iteratively-applied version of this framework could ultimately enable a robot to continuously improve its feature representations, thereby automatically becoming more and more robust in mapping, recognizing, and tracking its environment and the objects and people therein. While this chapter is essentially a proof-of-concept, the next chapter will explore ways to apply this approach to larger scale.

Additional Resources

- Code for the custom caffe layers can be found here: https://github.com/tschmidt23/caffe
- A video showing additional results can be found here: https://youtu.be/jfXyAypAQWk

Chapter 7

Correspondence Estimation in Large Naturally Evolving Scenes

In the previous chapter, we further advanced the idea (sketched out in Chapter 4) that generative model-based vision techniques can be used to automate the process of labeling data for the purpose of training deep neural networks. As a proof of concept, we showed that dense reconstruction techniques can be used to supervise deep networks that produce dense descriptors of images which can then be used to align models of the same object or relocalize a model in a new observation. We were not able to provide inter-sequence labels automatically (i.e. we did not know correspondences between frames in different videos), but we showed that these labels aren't necessary—the network learns to produce descriptors which allow for identifying inter-sequence correspondences regardless.

The experiments in the last section were somewhat limited in scale, however. The human dataset comprises videos of only a single person, and while the model does deform and lighting varies drastically, it is nevertheless not too challenging to visually separate foreground from background. Furthermore, the scope of the appearance model the network must learn is limited — as long as a network can learn the appearance of the head and shoulders it can solve this task well. The 7-scenes dataset might appear to be much more comprehensive, and the variety of materials and objects in all seven scenes does necessitate learning a broader appearance model, but a few factors make localization in these scenes easier than it seems. First, there is little to no lighting variation across multiple videos of the same scene, meaning a deep network can easily (and likely will) overfit to illumination-dependent features in the images. Second, the reconstructions tend to focus on only small parts of each scene with limited viewpoint variability (likely due to loop closure challenges when reconstructing the scenes). Furthermore, the scenes are entirely rigid, meaning that it is not actually necessary for the network to learn a strong representation of everything in the scene—robust recognition of a few objects or key points is sufficient for highly accurate camera pose estimation[1]. Needless to say, if one were to change the lighting or significantly rearrange objects in these scenes, networks trained on the 7-scenes dataset could struggle to recognize the images.

[1] As a further simplification in the 7-scenes dataset, the standard testing procedure is to localize the camera pose *assuming the scene is known*, i.e. it is not necessary to also estimate which of the scenes is being observed. This means there is no need to be able to differentiate between the multiple scenes, which is why scene coordinate regression works on this dataset.

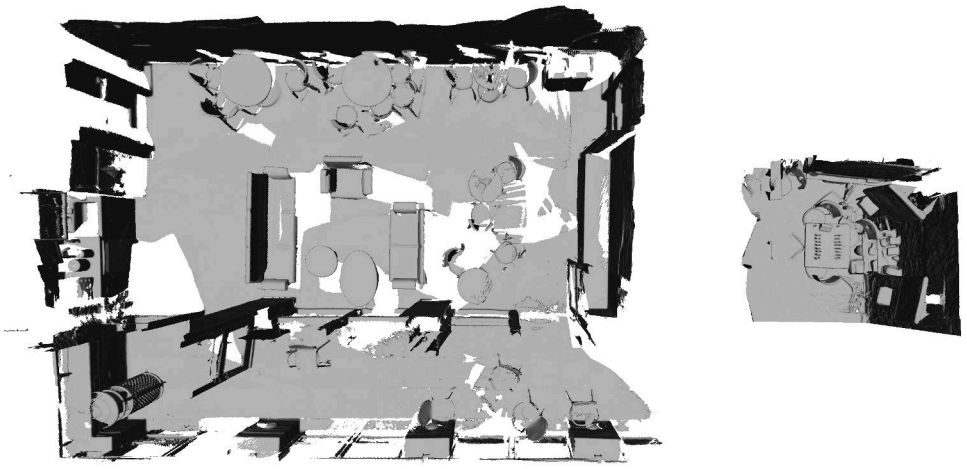


FIGURE 7.1: A side-by-side comparison showing an overhead view of one of the larger meshes from the Jaech Gallery dataset (left) next to the 'chess' scene from the 7-scenes dataset (right). In addition to the smaller scale, note that the 'chess' scene only includes two walls of the room.

In Chapter 1, we discussed demands on embodied vision systems, a few of which are highly relevant here. For one, an embodied agent often can't afford to make a distinction between foreground and background as things appearing in what might otherwise be considered the background could in fact be relevant. In other words, everything is foreground. In addition, an embodied vision system needs information that is both detailed and highly accurate in any imaging scenario the agent might encounter, which could include large variations in lighting conditions.

In keeping with the depth-first approach informed by embodied applications, this chapter introduces a new dataset designed to test the ability of an embodied agent to learn an extremely robust appearance model for a single complex and dynamic scene, ideally with little to no supervision. To this end, we combine the challenging variability in illumination seen in the human dataset from Chapter 6 with the larger scale and density of the 7-scenes dataset. In contrast to the 7-scenes dataset in which segmentation is not needed due to scene rigidity and the dataset from Chapter 6 in which segmentation is given implicitly by pointing the camera at the subject, we here additionally focus on a scene where multiple objects are frequently moved between visitations. Our goal, as before, is to learn dense descriptors as in Chapter which allow for correspondence estimation despite changes to illumination, viewpoint, and scene reconfiguration, without any labels regarding global identity or object membership.

7.1 The Jaech Gallery Dataset

The subject of this dataset is the Jaech Gallery at the Paul G. Allen Center for Computer Science and Engineering at the University of Washington. It was chosen for a variety of reasons. First, it is a multi-purpose room and therefore contains many elements that are typical of more specific classes of rooms. For example, there is a refrigerator, microwave,

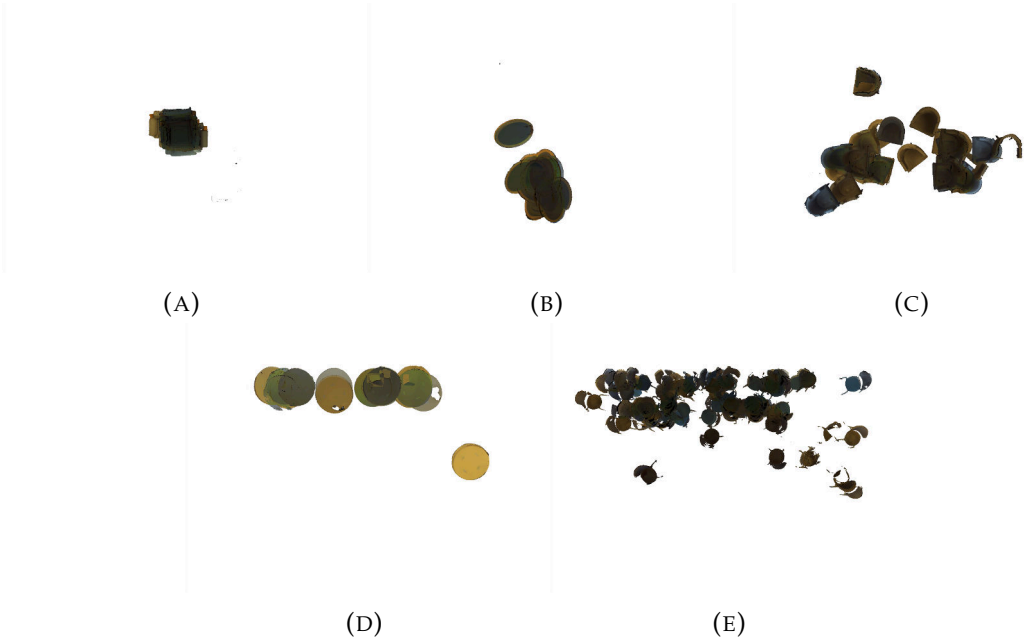


FIGURE 7.2: A visualization of the distribution of the location of various pieces of furniture in the Jaech Gallery dataset. Manually segmented object model renderings are superimposed on one another according to a registration of the static components of the room. Note that some items of furniture, such as the armchair (A) and side table (B) are typically in more or less the same spot, while other chairs (C, E) and the tables (D) are observed in a wider variety of locations and orientations.

dishwasher, and sink as one might find in a typical kitchen, there are couches and side tables as one might find in a typical living room, and there are chairs and tables as one might find in a typical study.

Another appealing aspect of the Jaech Gallery is that one of its longest walls consists almost entirely of windows. Thus, the appearance of the room and objects contained within changes drastically throughout the day and with differing weather conditions, allowing us to test for robustness to illumination changes.

Finally, Jaech Gallery is a highly frequented room and therefore experiences dynamic appearance changes due to constant intervention from humans. Although all images were captured while the room was empty, people visited the room between captures and induced changes by, for example, erasing or adding new content to the white board or re-arranging furniture. Other than propping open doors or drawing the blinds to manage lighting conditions, the room was always captured as-is, meaning the state of objects in the room follows the extremely complicated distribution of the real world, which a robot operating in that environment would face. This distribution is visualized for select objects in Figure 7.2.

While this dataset only contains images of a single room, we believe it is still a useful benchmark for embodied vision tasks. After all, one can imagine robotics tasks such as elder care which might take place entirely in a single room. In such scenarios, it is

much more important for the robot to have highly accurate perception of that environment to have a highly general (but less accurate) vision system. Furthermore, although the dataset does contain only a single room, it is relatively large in terms of physical scale. For a size comparison between the Jaech Gallery and the 'chess' scene from the 7-scenes dataset, see Figure 7.1. In addition to the much larger physical scale, the Jaech Gallery reconstructions cover all four walls of the room and thus contain a high degree of viewpoint variability, unlike 7-scenes.

Other datasets, such as those by Valentin et al., (2016) and Chang et al., (2017), instead provide a single reconstruction of many different environments. While this is likely more useful for training local, SIFT-like descriptors that exhibit strong generalization performance, it is nevertheless an approach more aligned with the goals of web-scale vision. From an embodied perspective, we are more interested in generalizing to different lighting conditions and room configurations such that a robot can successfully be deployed in a constrained environment before attempting to generalize to any possible environment. Even if one were to learn an appearance model using a dataset with larger scope and more variation such as those cited above, it would still make sense to fine-tune on a more specific dataset such as the one presented in this chapter before deploying a robot.

7.1.1 Data Capture

Some statistics of the Jaech Gallery Dataset are collected in Table 7.1. One unique aspect of this new dataset is the quality of the images. Data was captured using a structured light sensor, which provides depth maps for dense reconstruction, as well as a rigidly attached high-definition color sensor with a global shutter. The high quality of the resulting color images as compared to those provided by the color sensor packaged with the depth sensor helped improve the quality of the reconstructions by allowing for more precise feature-based tracking, but also provides higher-resolution images for input to deep networks.

These higher quality images are also much more realistic in an embodied vision setting. Higher resolution images carry more information about, for example, precise segmentation boundaries and subtle texture cues. This also poses a further challenge for learning methods, which must balance the benefits of extracting fine-grained information from high resolution imagery the practical constraints on memory and computation.

7.1.2 Automated Labeling

As in prior work, the data is labeled by generating a vertex map for each frame of each video identifying a model vertex associated with each pixel. This is generated by first creating a dense reconstruction of the scene, then rendering the reconstruction according to the estimated pose from which each color image was captured. Given these vertex maps, pixels that observe the same model vertex can then be assumed to be in correspondence. Again, as before, the reconstruction for each video is completely separate, such that correspondences cannot be automatically identified across videos.

However, in prior work, models were reconstructed using the depth stream of an RGB-D camera, and the trajectory of the color camera was estimated using an extrinsic calibration and by assuming depth and color frames were captured simultaneously.

Number of sequences	59
Total number of frames	221,417
Min sequence length	1,646 frames
Max sequence length	6,774 frames
Depth Shutter	Rolling
Depth Resolution	640×480
Color Shutter	Global
Color Resolution	1920×1200
Min vertex count	796 K
Max vertex count	1,915 K
Mean vertex count	1,257 K
Min scene diameter	9.04 m
Max scene diameter	12.39 m

TABLE 7.1: Statistics of the Jaech Gallery dataset. The sequence lengths given pertain to the color stream.

This is limiting, however, because tracking and reconstruction from depth alone is only possible when each depth frame contains sufficient structure within about three meters of the sensor. The Jaech Gallery is more than ten meters across on its diagonal, and therefore there are many meaningful images of the room that can be captured in which all or most observed surfaces are more than three meters distant. Thus, for this dataset, the color trajectory is estimated first using a sparse SLAM system, and then the depth trajectory is fit to the color trajectory and sparse map offline via global bundle adjustment.

More specifically, the data processing process is as follows:

1. An initial estimate of the color trajectory is estimated using ORB-SLAM (Mur-Artal and Tardós, 2017). In the rare cases where tracking failed, the video was run in reverse. This approach produced a reasonable color trajectory for almost all videos.

2. An initial estimate of the scale offset between the sparse and dense maps was provided manually (this was the only manual step in the process, and could be automated).

3. A global bundle adjustment optimizes the 3D position of the sparse map points, the camera pose of color and depth keyframes, and the time offset between the timestamps recorded by the color and depth cameras. The objective minimized is a combination of a reprojective error for observations of map points in color keyframes (compared to 2D detections), a point-plane error for observations of map points in depth keyframes, a frame-to-frame point-plane ICP objective for overlapping depth keyframes, and finally an objective encouraging alignment of color and depth trajectories (c.f. section 7.1.3 for details).

4. After global bundle adjustment, the poses of frames that fall between keyframes are optimized against the keyframes.

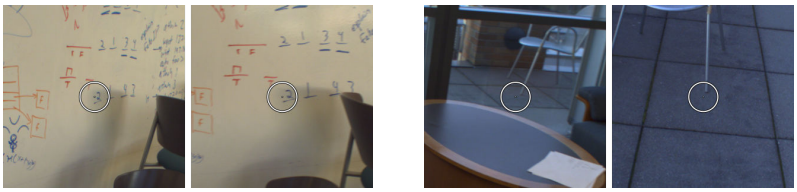


FIGURE 7.3: Example point correspondences identified by reprojection of a vertex from the reconstructed scene model into two images.

5. A 3D model is reconstructed using volumetric fusion (Curless and Levoy, 1996) of all depth frames in the video. A sparse voxel hashing scheme is used to fit the relatively large maps in memory at a satisfactory resolution.

6. A color is estimated for each vertex in the reconstruction using a geometric median to remove outliers. This is done by rendering the model given each pose on the color trajectory, and wherever a vertex is observed, the color at that pixel location in the corresponding image is added to the list of all colors observed for that vertex. Then a geometric median of the observed colors for each vertex is computed to remove outliers.

Some reconstructions resulting from this process are visualized in the left column of Figure 7.8. The reconstructions are by no means perfect, but most of the remaining errors are due to missing or incorrect data association (a further hypothesis to be tested is that learning descriptors from this dataset will allow more robust estimation of the correspondences within the training videos themselves and therefore allow for more accurate reconstructions). In any case, the reconstructions are of sufficient quality to establish dense correspondences between frames of the same video that are mostly correct. Figure 7.3 shows examples of correspondences estimated by projecting a single model vertex into two video frames.

7.1.3 Global Bundle Adjustment

This section gives a more formal and detailed explanation of step 3 above. The total energy function is a sum of four terms given by

$$E(\theta) = E_{\text{mvs}}(\theta) + \lambda_{\text{d2d}} E_{\text{d2d}}(\theta) + \lambda_{\text{s2d}} E_{\text{s2d}}(\theta) + \lambda_{\text{traj}} E_{\text{traj}}(\theta) \,, \tag{7.1}$$

where θ is the full set of variables being optimized, as given in Table 7.2, and each λ is a weight used to balance the relative contribution of the different objectives.

The first term, $E_{\text{mvs}}(\theta)$ is a standard multi-view stereo term involving the color poses $T_{w,c}$, sparse model points X, and a set of associated observations $o_k^i \in \Omega_c$ of sparse map point k in frame i, where Ω_c is the image plane of the color sensor. For each such observation, the loss is given by

$$e_{\text{mvs}}(\theta) = c\big(o_k^i - \pi(T_{w,c}^i * \mathbf{x_k})\big) \,, \tag{7.2}$$

where π is the non-linear projection function based on a calibrated camera model, and $c(\cdot)$ is a robust cost function (we used a Huber penalty). The small e denotes that this is

$T^i_{c,w}$	Transform from world to color camera coordinates for video frame i
$T^i_{d,w}$	Transform from world to depth camera coordinates for video frame i
t_{off}	Time offset between color and depth timestamps
t_{skew}	Time skew between color and depth timestamps
$\mathbf{x_k}$	Sparse model point k in world coordinates

TABLE 7.2: Variables Optimized in Bundle Adjustment

an individual error; E_{mvs} is simply equation 7.2 summed over all keypoint observations. We'll follow the same pattern with the other losses.

The second term, E_{d2d} is a standard dense point-plane reprojective frame-to-frame error defined for pairs depth maps $D^i(\mathbf{u})$ and $D^j(\mathbf{u})$. Each depth map has a corresponding normal map $N(\mathbf{u})$ and back-projected vertex map $V(\mathbf{u})$. The energy function for each pair is

$$e^{i,j}_{\text{d2d}}(\theta) = \sum_{\mathbf{u}\in\Omega_d} N^j(\mathbf{u}') \cdot \left(V^j(\mathbf{u}') - T^j_{d,w} * T^i_{w,d} * V^i(\mathbf{u})\right) , \tag{7.3}$$

where u' is the pixel on which the vertex in frame i projects in frame j, i.e.

$$u' = \pi(T^j_{d,w} * T^i_{w,d} * V^i(\mathbf{u}) . \tag{7.4}$$

We use standard checks on the validity of both depth measurements, normal agreement, and enforcement of a reasonable minimum and maximum depth.

The next term measures agreement between the sparse map and the depth measurement. It is given by

$$e^{i,k}_{\text{s2d}}(\theta) = c\left(N^i(\mathbf{u}') \cdot \left(V^j(\mathbf{u}') - T^i_{d,w} * \mathbf{x_k}\right)\right) , \tag{7.5}$$

where similar to the above we have

$$u' = \pi(T^i_{d,w} * \mathbf{x_k}) . \tag{7.6}$$

The final error term is perhaps the trickiest and ensures agreement between color and depth trajectories. This was required because the cameras were not time synchronized. First, we define the time according to the 'depth clock' relative to time according to the 'color clock', denoted by t_d and t_c, respectively, as

$$t_d = t_{\text{off}} + t_{\text{skew}} * t_c . \tag{7.7}$$

We'll also define $floor\_frame(\cdot)$ as the function which takes a color clock time and maps it to the index of the last color frame that arrived before this time. Given the fixed extrinsic calibration $T_{d,c}$, we can define the trajectory loss for depth frame i as

$$e^i_{\text{traj}}(\theta) = ||log(T^i_{w,d} * T_{d,c} * interp(T^j_{c,w}, T^{j+1}_{c,w}, t^i_d - t^j_c)||_W , \tag{7.8}$$

where $j = floor\_frame(t^i_d)$ and $interp(\cdot)$ is a function which interpolates between transformations in $\mathbb{SE}(3)$. Interpolation is done linearly on the $\mathfrak{se}(3)$ manifold. Finally, W is a weight matrix which sets the scale of the penalty between rotations and translations.

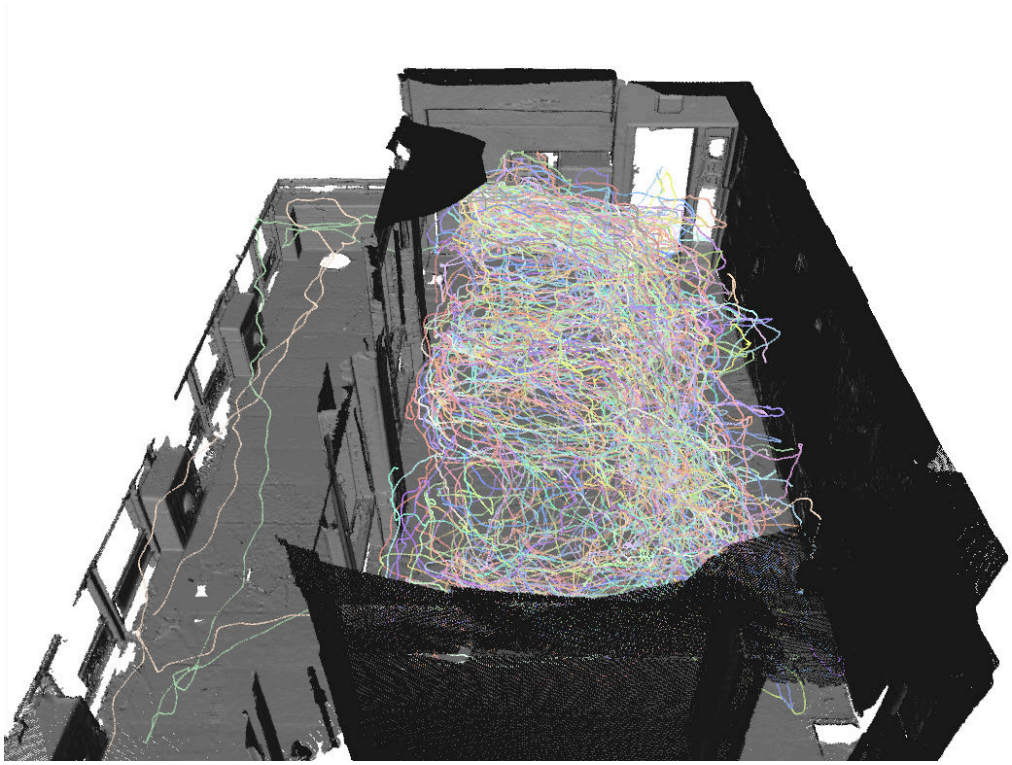


FIGURE 7.4: A visualization of the camera trajectories for all sequences in the Jaech Gallery Dataset. The vertices correspond to static parts of the scene, i.e. geometry that is consistent across all visitations. Each camera trajectory is shown using a different random color. Note the relatively dense coverage of the scene, excepting the patio area which was only traversed in two sequences.

All of these energy functions are differentiable, and thus the full error term is minimized using Gauss-Newton.

7.1.4 Rigid Map and Trajectory Alignment

A key concept of the line of research carried out in this and the previous chapter is that multiple observations of the same object at different times do not need to be registered in order to learn an appearance model of the object that is consistent across observations. However, because in this chapter we are introducing a dataset that may prove useful as training data or otherwise for other applications, the rigid alignment of the static components of the scene was also computed and is provided for future use. The process for computing this alignment was as follows:

1. Use ICP to compute a pairwise rigid alignment for every pair of mesh models and score it based on the number and average ICP error of inliers.

2. Compute a minimum spanning tree of the graph of all pairwise alignments, where edge weights are given by the ICP score.

3. Chain transformations in the tree to establish the pose of each mesh in relation to any other, which is unique given the tree structure.

Figure 7.4 visualizes the results by superimposing the camera trajectories from each visitation to Jaech Gallery on a point cloud model of the static components of the scene.

7.1.5 Object Discovery and Dynamic Scene Understanding

A key novelty of the Jaech Gallery Dataset is that it depicts many objects in the same scene that is visited repeatedly, with objects being reconfigured between visits. It is thus an interesting dataset for benchmarking the ability of a system to discover objects that have been observed multiple times, and to identify the presence and pose of these objects in new images.

In this sense, this dataset is similar to the data used by Fehr et al., (2017), who demonstrated a state of the art solution to the object discovery problem without using learning. They simply rigidly aligned each reconstruction of the scene, then used the TSDF volumes to do space carving, allowing for the identification of the static components of the scene. They also fused dynamic components of the scene into object models using the approach of Furrer et al., (2018). The results are compelling, but the assumptions are limiting. This approach is fundamentally dependent on the quality of the reconstructions, but in challenging and large environments such as the Jaech Gallery it is challenging to achieve full global metric accuracy such that a single rigid transform aligns the TSDF volumes computed during multiple visitations to the scene. Furthermore, the TSDF carving approach is not well suited to cases in which objects move but only slightly so as to never fully vacate their previous locations and allow for positive TSDF observations. Karpathy, Miller, and Fei-Fei, (2013) showed how this limitation might be avoided by estimating entities that may constitute objects without requiring that they move.

Similar work was carried out by Herbst et al., (2011) and Herbst, Ren, and Fox, (2011). They also assume that rigid alignments of multiple models of the same scene are available, but also incorporate appearance features into the segmentation of dynamic elements. This work predates the surge in deep learning in the computer vision literature, and thus more traditional features are used. The problem of segmenting dynamic from static scene content has also been explored by Finman et al., (2013) and Ambruş et al., (2014). The techniques they present for identifying and maintaining static maps and object databases are still relevant in this context, but this chapter presents a novel way to learn an invariant appearance model that could aid in the identification and reindentification of objects that move within a scene. This will be done by applying the self-supervised approach presented in 6, with a modification to allow it to scale more efficiently. This is the topic of the next section.

7.2 Dense Descriptor Learning with Proxies

In all of the datasets used in Chapter 6, most of the model of interest was visible in every frame. This is because the models are relatively small and the viewpoint variability is limited (in the human dataset, the person almost always faces the camera, and in the 7-scenes dataset the camera is typically always pointed towards the same part of the scene). When most of the model is visible in every frame, each observed frame provides a training signal concerning the representation of most of the model vertices. Furthermore, one can simply sample non-corresponding pixels at random from a single paired

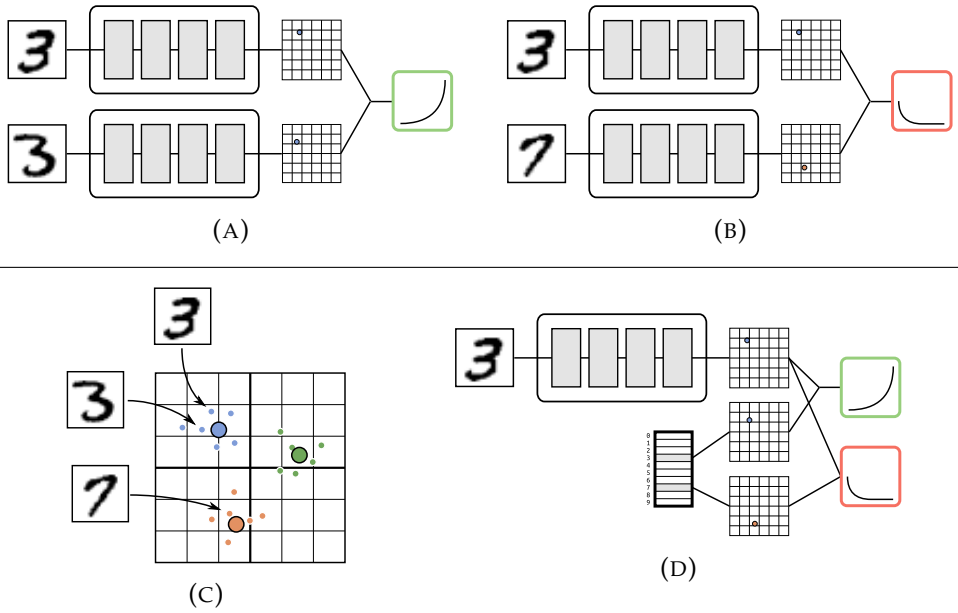


FIGURE 7.5: With a contrastive loss, a pair of images is passed through the network and an L2 or hinge loss is applied depending on whether they are of the same class (A) or different classes (B). This leads to clustering in the embedded space, visualized in (C) with images shown as points and colored according to class. With the contrastive proxy loss, a 'proxy' for each class is tracked, visualized in (C) as the larger points (one for each class). Then, a single image is passed through the network and compared against any number of stored proxies, as shown in (D). Gradients of the loss flow back to the proxies in addition to back-propagating through the network.

image and expect a fairly representative sample of all other descriptors the network might produce for *other* parts of the model.

With the Jaech Gallery Dataset, however, it is no longer the case that most of the model can be seen in every frame. This is due to both the increased scale of the scene and to the fact that the camera views the scene from a wide range of viewpoints and scales as it traversed the scene (in contrast to the 7-scenes dataset). This causes two problems: first, each image only provides a training signal concerning the representation of one (often relatively small) portion of the model, and second, a representative sample of non-corresponding pixel descriptors would require a large number of images to be processed by the network. The latter problem could be solved by increasing the batch size, but in practice the large memory footprint of the network limits the batch size to just a few images. The former problem makes it difficult for the network to even keep track of what it has learned so far, as many gradient steps may be taken in between observations of the same point on the model during training. Furthermore, it is also not the case that the entire network output for a given input image can be used for matching, as the difference in viewpoint results in only partial overlap between image pairs.

One possible solution to this problem from the deep metric learning literature is the use of 'proxies' as described by Zhang et al., (2017). A proxy is defined in this context

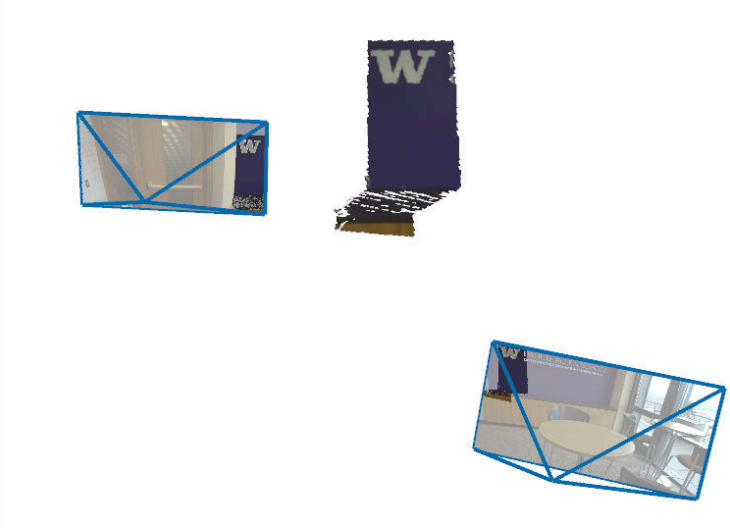


FIGURE 7.6: When training deep semantic segmentation networks it is common to apply a pixel-wise loss, thus providing a training signal across the entire network output. When training dense descriptor networks with contrastive loss, it is not possible to make such efficient use of the network output due to partial overlap between image pairs. This figure illustrates this problem using a pair of images rendered along with a camera frustum according to the pose from which they were captured. The portion of the map that was viewed in *both* images is also shown, and pixels on which this portion of the map project are highlighted in each image. Note that well under half of the pixels in each image overlap on the model. Such frame pairs are common in the dataset, and simply discarding pairs such as this is not an acceptable solution as it is exactly the pairs with large viewpoint and scale disparities that provide the strong training signal that leads to invariant descriptors.

as a descriptor which serves as an exemplar or representative of a set of images, in the sense that if any of the represented images were passed through the corresponding deep feature embedding network the resulting feature vectors would all be near (and ideally centered on) the proxy. One typical use case is to define one proxy per image class, in which case the proxies can be thought of as the abstract internal representation of the class that the network is currently using. Equipped with proxies, one can re-formulate a loss function over two or more descriptors and replace all but one of them with proxy representations. The gradients of the loss function can backpropagate through the network to change the representation of the image (as usual), but they can also be used to update the proxies directly without requiring any interaction with the network.

Zhang et al., (2017) used proxies in combination with a triplet loss, citing the scale-invariance property of descriptors learned with the triplet loss as motivation for this design decision. With our focus on fast and efficient point matching, we opt for a pair-wise contrastive loss instead. In other related work, Wen et al., (2016) used a contrastive loss with class proxies to regularize a network that was also supervised by a softmax classification loss. In this case, the proxies were simply running averages of network output for each class, but we'll combine the contrastive loss formulation of Wei et al., (2016) with the proxy learning technique used by Zhang et al., (2017) to formulate our contrastive proxy loss.

More formally, assume for each class i we have a proxy $\mathbf{p_i} \in \mathbb{R}^D$, where D is again

the dimensionality of the descriptor. We can then reformulate the loss defined in equation 6.1:

$$L(I_a, \mathbf{u_a}, b) = \begin{cases} d(I_a, \mathbf{u_a}, \mathbf{p_b})^2 & X_a(\mathbf{u_a}) = b) \\ \max(0, M - d(I_a, \mathbf{u_a}, \mathbf{p_b}))^2 & \text{otherwise} \end{cases}, \tag{7.9}$$

where the distance function is now given by

$$d(I_a, \mathbf{u_a}, \mathbf{p_b})) = ||f(I_a)(\mathbf{u_a}) - \mathbf{p_b}||_2 \,. \tag{7.10}$$

Note that this loss only depends on a single image I_a due to the use of a proxy. This concept is illustrated in Figure 7.5.

In our case, we don't exactly have 'classes' per se. However, we can simply associate the projected vertex value at each pixel location in each image with the nearest discrete vertex in the mesh, and then treat all of these observations as a 'class'. To this end, we then associate a proxy vector of length D with each map vertex.

There are a number of advantages to deep metric learning with proxies, some of them specific to our application and others more general. More generally, Zhang et al., (2017) showed empirically that the proxy triplet loss leads to much faster convergence than a standard triplet loss. This is likely due to improved sample complexity (because the loss is defined over individual images rather than pairs of images) and a decrease in the variance of the gradients when training with proxies. As for our specific scenario, proxies help address some of the challenges discussed above. Even if many gradient steps occur between observations of some particular model vertex, the proxies serve as a sort of external memory for the network which allows it to develop persistent representations. Also, sampling a representative sample of non-corresponding pixels is trivial because we can simply sample vertices uniformly at random from the model. As an added benefit, the sampling of corresponding pixels under the contrastive proxy loss also allows us to make much more efficient use of the network because every (valid) pixel can be compared with its stored proxy representation. In contrast, applying the pair-image training procedure described in section 6.2.2 often means that that only a relatively small fraction of pixels in one or both images have any corresponding pixel in the other image due to changes in viewpoint, scale, or both, and these cases cannot be ignored if the goal is to learn strong viewpoint and scale invariance. This problem is illustrated in Figure 7.6.

7.2.1 Training Details

We again used an FCN model to produce our dense descriptors, but modified slightly to handle higher-resolution input. We input images to the network at the full 1920×1200 resolution as captured by the sensor. In order to save memory and because having an equal output resolution is not strictly necessary, we replace the first pooling operation in the network, which was originally 2×2 max pooling with stride 2, with a 5×5 max pooling with stride 5. Then, when upsampling with deconvolutions, we keep the original 2x upsampling rate, yielding an overall resolution reduction of 2.5x such that the feature map resolution is 768×480 (similar to the size of the networks used in Chapter 6, with a slightly different aspect ratio).

For each pixel in each output feature map, we consult the associated rendered vertex map and fetch the associated proxy. This proxy is paired with the network output for

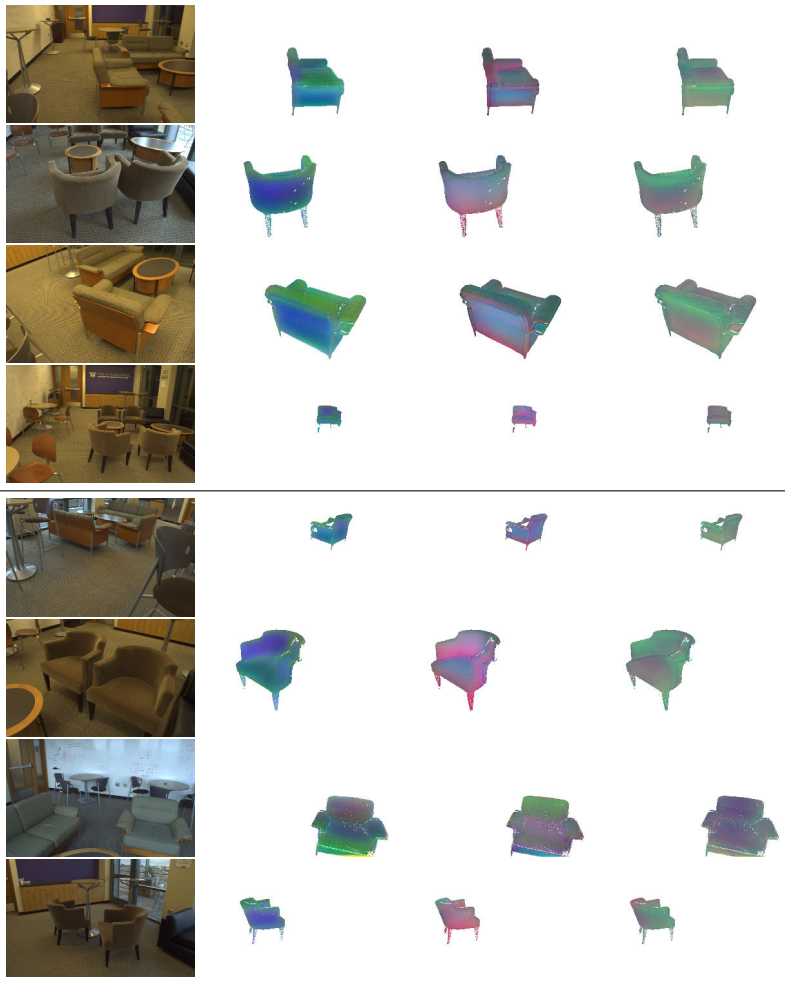


FIGURE 7.7: This figure shows the network output for a selection of frames from the Jaech Gallery dataset. To ease the interpretation, we masked the dense descriptors to focus on a particular object (based on manual segmentation). The learned features are visualized as in the right column of Figure 6.4, but in this case the descriptors are 8-dimensional and thus require three images to display rather than one. The first feature image maps the values of the first three dimensions of each descriptor to the red, green, and blue color channels, the second image does the same for the fourth through sixth dimensions, and the final image shows the last two dimensions in red and green with the blue channel fixed at an intermediate value. A pair of descriptor values that is similar thus appears with a similar color in all three images, but a similar color in one of the images does not necessarily imply descriptor similarity. Interestingly, the network learns similar representations for both armchairs, an effect also observed by Florence, Manuelli, and Tedrake, (2018). While this may not be desirable as it could lead to confusion between the two chair types, it likely occurs because both chairs were never in the room at the same time.

that pixel and a loss is computed according to the first case in equation 7.9 (i.e. Euclidean loss). We also select for each pixel two other proxy vectors uniformly at random from the same model, and compute a loss according to the second case in equation 7.9 (i.e. Hinge loss). The two losses are again balanced so that the contribution from corresponding

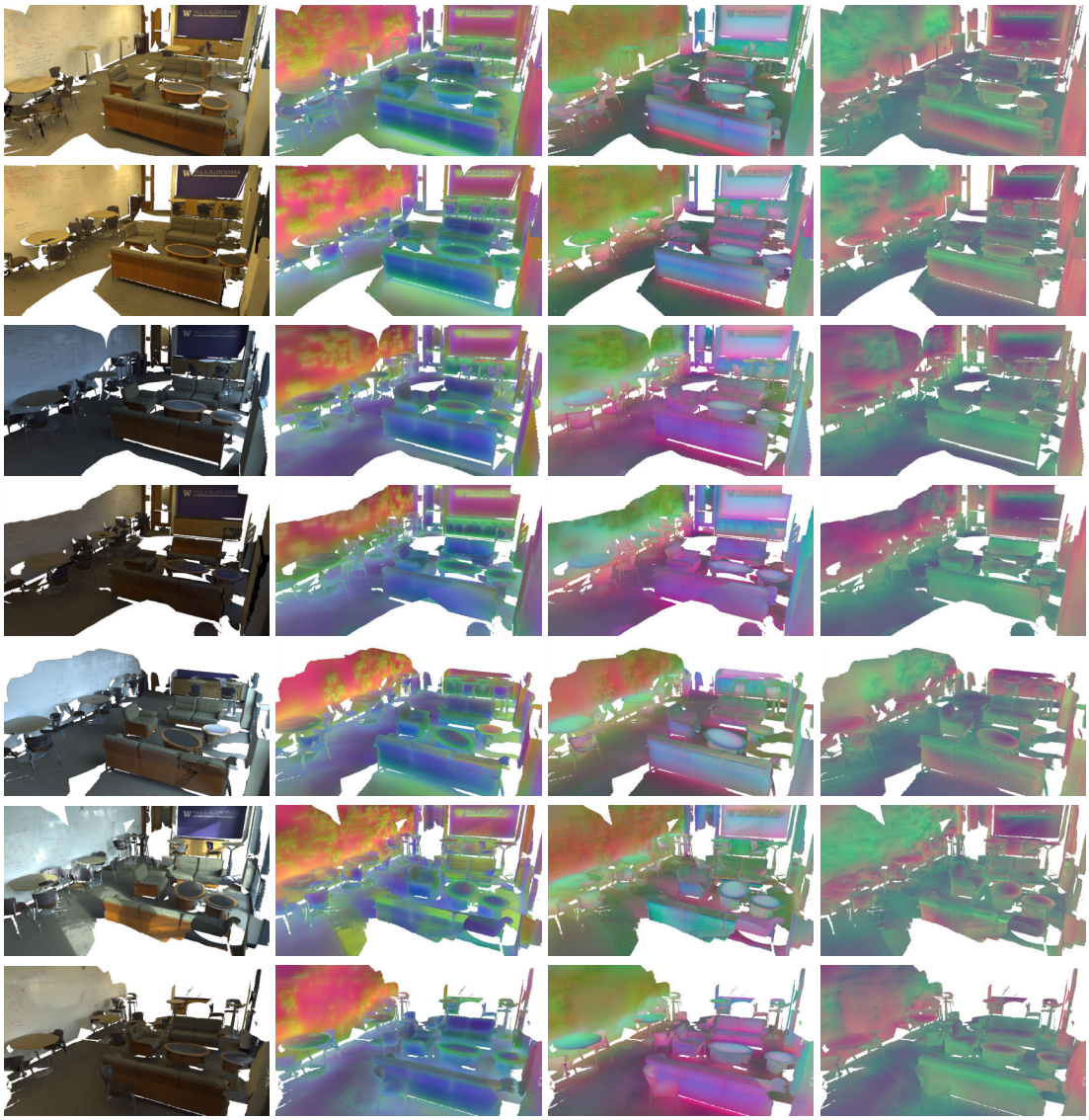


FIGURE 7.8: A visualization of the Jaech Gallery dataset and the result of self-supervised visual descriptor learning on the dataset. The left column shows a rendering of a textured meshed reconstructed from one video sequence as described in section 7.1.2. The right column shows the same mesh from the same viewpoint, but with each vertex colored according to the average descriptor value for all observations of that vertex and visualized as in Figure 7.7,in this case without masking (figure continues on the next page).

and non-corresponding pairings is equal.

During the backwards pass, the partial derivatives of the loss with respect to the proxies selected for each pixel are also computed. All resulting gradients associated with the same proxy are then pooled and sent to the proxy database to update the stored values.

The dataset is divided into a training set (50 sequences) and a held-out validation set (9 sequences). As mentioned, each vertex in the models associated with each training video was assigned a proxy descriptor value, which was initialized by sampling uniformly at random from a mean-zero Gaussian distribution with standard deviation 0.05. The network is optimized with AdaGrad and a learning rate of 0.0001 (due to the small batch size of 2 images, which is all that could fit in memory on a single GPU). However, we found empirically that it is important for the proxies to be able to adapt quickly relative to the network parameters, and therefore proxies are optimized (also using AdaGrad) with a learning rate of 1. Descriptors were of dimensionality $D = 8$.

In order to make the network more invariant to illumination-dependent hue in the input images, we applied a randomized gamma correction for each image every time it is sampled for training. A different gamma value is sampled for the red, green, and blue channels, which results in non-linear intensity changes and potentially a hue shift as well if the sampled values differ across channels.

7.3 Results

As in the prior experiments, the network is again capable of learning dense descriptors that are for the most part invariant to viewpoint, illumination and pose, and largely consistent across separate visitations to the scene despite not being explicitly trained to do so. Due to the additional focus on objects that move within the scene, the consistency of learned objects features is visualized in Figure 7.7, in which the features have been masked based on a manual segmentation in order to ease comparison. Despite large changes in viewpoint and scale, corresponding points on these objects are represented with similar features. As before, there was no supervisory signal encouraging the network to learn similar features for the same object in separate sequences, as the models were not registered. The fact that similar features result is even more remarkable in this context as there are many objects moving independently, and the network is not only given no registration information but also no information regarding the segmentation of the scene into separate objects.

Figure 7.8 gives a more thorough overview of how the network learns to represent this scene. It is analogous to Figure 6.4, but with additional feature renderings due to the higher dimensionality of the descriptors. As compared to Figure 6.4, note that, because the scene is much larger than a person, there is a lot more for the network to learn and it has indeed succeeded at separating out many visual concepts.

7.3.1 Quantitative Evaluations

We performed the same quantitative experiment described in section 6.3.2 in order to evaluate the quality of the learned features and to quantify the advantages gained using the contrastive proxy loss as opposed to a pure proxy loss. That is, for a number of test cases, we compute the descriptor for a query location in one image and then compute

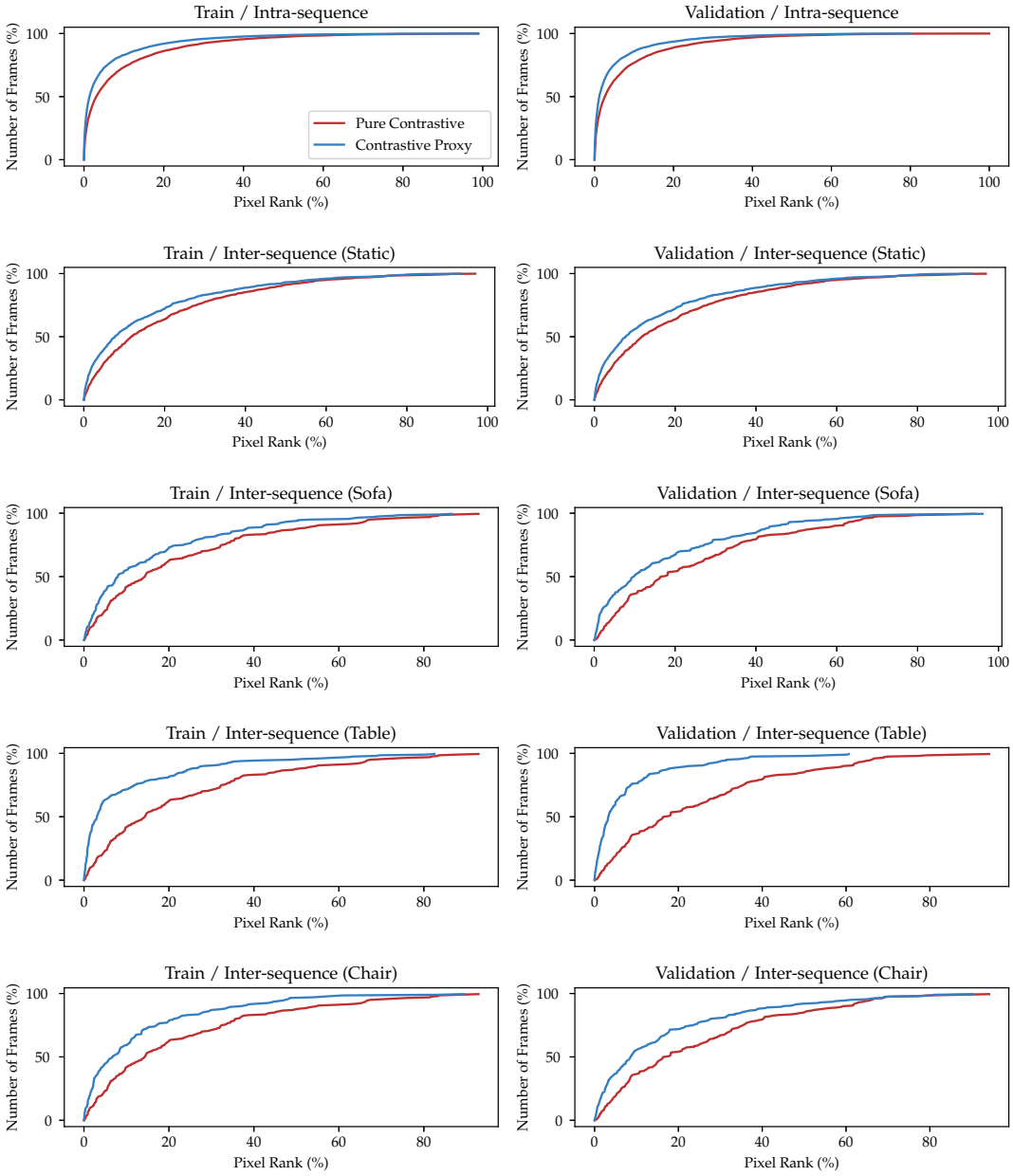


FIGURE 7.9: Quantitative results of self-supervised descriptor learning on the Jaech Gallery dataset, comparing the contrastive proxy loss with the pure contrastive loss. The evaluation follows the procedure of 6.3.2. The row shows results for intra-sequence samples. The bottom four rows show results for inter-sequence samples based on manual segmentation of the static scene and three different objects, follow by automated rigid alignment. Example correspondences for the bottom four rows can be seen in Figure 7.10.

FIGURE 7.10: A visualization of learned feature proximity using heat maps. For each row, the left column shows the query image with the query point indicated. The right column shows the target image combined with a heat map indicating the distance in feature space from the query point and every pixel in the target feature map (red is closer). The ground truth corresponding point is also indicated. The first row shows a correspondence from the static scene, and the bottom three rows show correspondences from the objects associated with the bottom three rows of Figure 7.9 in the same order.

the dense descriptor map for a target image. We sort pixels in the target image based on distance from the query point in descriptor space and record the rank of the ground truth correspondence.

The results are shown in Figure 7.9. As in Figure 6.6, plots in the left column are computed using training sequences and plots on the right column are computed using held-out validation sequences (i.e. entire sequences for which the network saw no frames during training). Each network processed 2 million images, roughly equivalent to ten epochs. It can be seen in the upper-left plot that the use of proxy loss leads to greater performance on the training objective. As seen in Chapter 6, the quantitative results are very similar for training and validation data, indicating that the density of labels and difficulty of the training objective result in minimal overfitting.

The first row of Figure 7.9 shows results for corresponding points in the same sequence, while all other rows shows results for corresponding points in different sequences. The second row shows results for points that belong to static components of the scene (walls, floor, and fixed furniture such as the refrigerator), and the remaining rows each show results for a single object that moves within the scene. For these points, the correspondence is provided by the rigid alignment described in 7.1.4 as well as a manually automated segmentation of the scene. Thus, while some of the decrease in performance when sampling corresponding points from different sequences is due to challenging appearance variation and object motion, some is also due to lower quality 'ground truth' labels. A few of these challenging inter-sequence test cases are visualized in Figure 7.10. Note that, despite the presence of extremely challenging inter-sequence dense correspondence queries, the ground truth correspondence ranks within the top 10% of pixels when sorted by distance in feature space for roughly half of all queries on both training and held-out validation sequences.

7.3.2 Object Relocalization Experiment

Due to the consistency of the learned descriptors, they can again be used in a relocalization scenario similar to the one shown in Figure 6.8. However, relocalization for this dataset includes an additional challenge because there actually a number of objects that can move completely independently. Whereas in Chapter 6 we always knew which model we were observing and only had to determine its pose, now we also must determine which models are present (not all objects appear in all scenes do to varying coverage and the fact that the furniture was changed in the middle of dataset collection). One easy solution to this problem would be to simply attempt to localize each object in the scene and apply some acceptance threshold above which the object is assumed to be present, thus determining both which objects are present based on some likelihood model and the poses of the objects. However, this would be expensive, and obviously does not scale as the number of objects increases.

Instead, we show how our learned descriptors can be used to both determine the objects in view and their poses in more or less real time. The first step is to generate a locality sensitive hashing function based on a random projection matrix $P \in \mathbb{R}^{B \times D}$, where B is the number of bits in the bucket index of the desired hashing function (we typically use $B = 64$. To hash a descriptor vector $\mathbf{v}$ we first compute $P\mathbf{v}$, then set each bit in the hash index to 1 or 0 depending on whether the corresponding index in $P\mathbf{v}$ is positive or negative.

FIGURE 7.11: Objects detected by the object localization system described in section 7.3.2 in one of the held-out sequences. The video was recorded at night under artificial illumination, but the system localizes objects that were observed under many different lighting conditions.

Next, we need a set of objects to look for. For this, we simply manually segmented a number of objects from meshes in which they were observed from many viewpoints, resulting in relatively complete object models. While in our case we did this manually, we did not supply any semantic labels as to what the objects were, and in any case this sort of segmentation could in theory be done automatically by a robot that is capable of poking objects in the scene (c.f. Chang, Smith, and Fox, (2012)). Furthermore, the segmentations were not used as a basis for any further training of the network, so all supervision used in this experiment was still auto-annotated.

For each of the segmented objects, we compute a histogram of all binary codes resulting from hashing the mean features associated with each vertex of the segmented object model. Then, for each new observed frame, we compute the dense feature map and similarly compute a histogram of all observed descriptors. We can then very quickly compute the histogram intersections (normalized by the number of points in the object) and, given the overlap, compute the relative probability that each object appears in the scene. We then sample 4 objects from this distribution to attempt to localize.

For localization, we follow a similar procedure to the one used in section 6.3.2. However, instead of preemptive RANSAC, we simply take the highest scoring out of 1024 pose hypothesis and refine it with point-plane ICP. Given the refined pose, we compute a score based on the geometric alignment as well as the descriptor agreement between all observed vertices on the model and the observation. If the score is sufficiently high, we accept the object relocalization and add it to the set of tracked objects.

Results of this localization procedure can be found in the accompanying videos. The database of objects that were provided for matching included multiple copies of objects as observed in multiple lighting conditions. While one might expect the system to match objects that were modeled under similar lighting conditions, this is not always the case. Figure 7.11 shows renderings of matched objects textured according to their source mesh models. The matched models exhibit a range of illumination-dependent appearances, but the fact that they successfully passed all thresholds for matching indicates a high level of similarity in the descriptor space, indicating that our learned descriptors are largely invariant to illumination changes. Furthermore, one of the accompanying videos demonstrates that the described object localization procedure can,

FIGURE 7.12: This figure shows the benefits of the dense semantic information encoded in the learned descriptors. At left, we show an image from a held-out validation sequence in which a table model has been matched. We then render the expected features according to the source map for the object and compare them to the actual features produced by the network for this observation, and color each pixel according to the agreement (blue is more similar). Feature mismatch is caused by occlusion by the chair and the towel, which could have been detected based on the depth map, but also by the paper on the table which does not appear in the depth map. This would be useful for a robot tasked with clearing the tables, for example.

at least in some cases, recognize these known objects even when they are observed outside of the Jaech Gallery.

Ultimately, this relocalization system is similar to the SLAM++ system developed by Salas-Moreno et al., (2013), which uses CAD models and point-pair features to recognize them using depth information only and simultaneously tracks the camera position relative to the detected objects. Our system is designed as a proof of concept for the utility of the learned descriptors and not necessarily meant to show improved detection performance over SLAM++. That being said, there are a few advantages to our appearance-based approach. One is illustrated in Figure 7.12, which shows that occluding objects that are too thin to meaningfully affect the depth map can still be detected using dense appearance-based descriptors. Furthermore, in this chapter we show relocalization of multiple rigid models, but as we demonstrated in Chapter 6 our descriptor learning technique applies to non-rigid models as well, whereas point-pair features are not deformation-invariant. Finally, many of the objects in this dataset are mostly planar surfaces without distinctive shape, and there are even pairs of objects with nearly identical shape such as the trash and recycling bins and the first aid kit and the exit sign. Our appearance-based features are thus critical in rejecting false positive matches for these objects which might not be possible with only shape information. In practice, it would be wise to combine both shape- and appearance-based features.

Discussion

In this chapter we adapted the self-supervised dense descriptor learning technique from the previous chapter to larger scale and more challenging problem. To do this, we collected a new dataset comprising 59 videos depicting a large student lounge under a wide variety of illumination conditions and with a number of different furniture configurations. In order to scale the learning technique, we also introduce a novel application

of a 'contrastive proxy loss' to the problem of self-supervised dense descriptor learning, accelerating convergence and allowing the network to learn representations of a number of objects without catastrophic forgetting. We were then able to use these descriptors to not only determine the pose of an object with no prior, as in Chapter 6, but also to determine which objects are visible in an image and are even worth attempting to localize.

While the use of proxies for accelerated descriptor learning may seem prohibitively expensive, it is not actually as bad as appears at first glance. Our descriptors are low-dimensional (compared to, e.g., SIFT), and although there are nearly 100 million vertices in the full training set the proxies easily fit in system memory. If this were not the case, an on-disk database with efficient caching could also be used. Furthermore, it is clear from Figure 7.8 that the scene is very much over-represented. One could imagine a training procedure with multiple phases in which training is intermittently paused so that indistinguishable vertices can be merged. This might be desirable because of visual similarity (i.e. one could merge proxies for nearby vertices on low-texture surfaces such as walls which are visual indistinguishable), or because the same surface appears in multiple maps (i.e. if a chair is observed in multiple maps, it only needs to be represented once, so if such a re-occurrence can be detected — with very high precision — the models can be aligned and the corresponding proxies can be merged). If the latter merging strategy is used, one can expect the cost of storing proxies to be linear with respect to the size of the observed environment rather than linear in the number of observations.

There are additional research uses of the Jaech Gallery dataset beyond what we have shown here. One could imagine applying a strategy similar to that used by Fehr et al., (2017) to do the alignment and merging of static and object models from different visitations. This would be a bit more challenging on this dataset, due to the clutter and the fact that some objects move but only slightly so that TSDF free-space carving would not remove them from a static map. On the other hand, it could be made easier by the fact that we have learned strong features on this particular scene that can be used to do alignment and scoring of matches (and, ideally, removal of clutter). Given a more complete, static mesh and complete object models, one could use the data to develop and test 3D scene completion systems which attempt to produce the full static and object models given the partial meshes shown in this chapter. Because a manual alignment of the static meshes is available and we have solved for the camera trajectories, the dataset could also be used to develop indoor relocalization systems and evaluate how well they handle extreme illumination changes and object rearrangement. Finally, given depth maps formed by rendering the mesh model, the dataset could also be used (likely in conjunction with other data) to train systems that solve tasks such as depth from monocular images or depth from multiview stereo, so long as some noise in the depth is tolerable.

This chapter represents a further step in our goal to leverage the combined strengths of generative model-based vision and discriminative learning-based vision to produce powerful vision systems with embodied vision goals in mind. As in Chapter 4, we are able to use generative model-based techniques to label training data for training deep networks to recognize and localize objects, but because we now use reconstruction techniques we do not need object models or any manual initialization — the network learns entirely from scratch.

Additional Resource

- The Jaech Gallery Dataset can be found here: https://drive.google.com/open?id=19E8MK9Wms3oVYr5D48BVM_QwrCEdYLL0.
- A video corollary to Figure 7.8 can be found here: https://youtu.be/5t64CSqAYxY.
- A video showing the results of the object relocalization experiment on the sequence "041" can be found here: https://youtu.be/KEw6nMFZvTw.
- A video showing the results of the object relocalization experiment on the sequence "053" can be found here: https://youtu.be/JX6iX__7gbQ.
- A video showing the object relocalization experiment detecting an object observed out-of-context can be found here: https://youtu.be/svcZao4chqE.

Chapter 8

Conclusion

If we want to see robots applied more successfully in more challenging and dynamic environments, we'll need powerful vision systems that allow these robots to accurately and efficiently perceive the world around them, ideally in a dense and highly granular fashion. These demands on robotic vision are for the most part best served by generative model-based techniques, at least given the current state of the art on discriminative, learning-based vision. However, many generative model-based vision techniques suffer from a lack of robustness due to their reliance on accurate estimates of observation-to-model correspondences and a lack of sophisticated correspondence estimation techniques which can provide such estimates in real-time. Discriminative vision might come to the rescue by learning appearance cues which can be used to estimate correspondence, but this typically requires large, labeled training sets.

In this dissertation, we explored both model-based and learning-based vision, beginning with the generic articulated model tracking system presented in Chapter 2. Chapter 3 applied this system in a robotics setting, where the detailed state estimates enabled automated grasp planning in challenging scenarios such as bi-manual manipulation, in which one of the robot's hands must grasp an object while it is held (and occluded) by the other hand. However, both of these tracking systems suffered from the aforementioned lack of robustness, such that poses had to be initialized manually or with an external system. Even after initialization, it is possible that tracking of the state can be lost if a new observation falls outside of the basin of convergence, which was explored in Chapter 5.

Beginning with Chapter 4, we began to explore the idea at the core of this dissertation, which is that discriminative models can help solve the remaining robustness problems in model-based vision, and furthermore that model-based vision can provide training data for learning-based vision. In this particular case, we used the tracking framework from Chapter 2 to accurately track the pose of an object in a video, allowing any frame from the video to be used to train a deep network (or other learning-based system) to recognize the pose of an object from a single image. This could have provided the initial state estimates needed in Chapter 3, but it still required 3D models of the objects to be provided and the tracking had to be initialized manually before pose labels could be generated.

In Chapter 6 we took this one step further by using dense reconstruction techniques to provide self-supervision. These systems do not require any model or initialization. The reconstructions were used to train networks which, rather than estimating pose directly, produce descriptors. These descriptors, in conjunction with a model, can then be used to estimate pose. The only drawback is that a fully separate model is generated

for each sequence, and if correspondences can't already be identified across models then these inter-sequence labels are not available for training (and if they can, then the descriptors are perhaps not needed). However, we showed that the network learns a consistent descriptor space regardless, such that the learned descriptors can be used to align models even when not explicitly trained to do so.

Finally, in Chapter 7, we scaled this approach up to a larger and more challenging dataset. Unlike in the preceding experiments, which tested whether a single model could be localized in an image that was already assumed to depict that particular model, the experiments in Chapter 7 tested the ability of the self-supervised descriptor learning approach to learn an appearance model for multiple objects which move independently between a large number of observations of the same scene, without any supervision on object segmentation. Our experiments showed that with this approach, we can not only estimate the poses of objects but also detect which objects are being observed, all using the same descriptors and without knowing anything about the objects during training.

8.1 Future Directions

While the work in this dissertation is a promising step towards more powerful embodied vision, there is still much work to be done. Further improvements are especially needed to speed up descriptor learning and further increase the scale of what can be learned without sacrificing on accuracy, density, or robustness. This final section will explore a few potential routes.

8.1.1 Better Network Architectures

In this dissertation we did not spend a lot of time discussing network architectures, as all of the networks used were borrowed from the dense semantic segmentation literature. However, the capabilities of the deep network is a crucial component of our self-supervised learning framework and advances in deep network architectures would lead to immediate improvements. There are still a lot of open questions in deep network architecture design, and future research will continue to improve on the state of the art. In particular, the applications in this dissertation would benefit from networks that are less prone to 'catastrophic forgetting' (Kirkpatrick et al., 2017). While the use of proxies in Chapter 7 led to faster convergence and allowed us to avoid situations in which the network forgot its representation for a subset of the models entirely, it's still possible that the network forgot how to map from inputs to the stored proxies even if the proxies were stored externally and explicitly. It is likely that architectural improvements and improved training algorithms could help remedy this.

8.1.2 Explicit Merging of Past Experiences

As alluded to at the end of Chapter 7, one interesting line of inquiry to follow up on the experiments with the Jaech Gallery Dataset would be to do explicit merging of multiple experiences of the same scene. This was in the context of a discussion on the efficiency of a proxy representation, but the concept applies more broadly — there is no need to remember what an object looked like every day for a year if its appearance does not change. If objects which appear repeatedly in this dataset can be identified, segmented,

and aligned, it would not only save memory by reducing storage requirements for the meshes and proxies, but it would also allow the network to use correspondence labels that cross the boundaries between sequences and thereby learn even more powerful, consistent, and discriminative representations. Perhaps even more useful is the ability to merge objects that are identical or at least visually indistinguishable even within a single reconstruction, such as two instances of the same chair or all of the points on the same textureless surface.

There a few key challenges here. One is the challenge of segmenting the objects, which is difficult to do in a consistent and principled way. For example, one might find that a fragment of one reconstruction representing a chair aligns well geometrically with a corresponding fragment from another reconstruction and that their features are in agreement as well, but it is not necessarily obvious that the object ends at the floor if they are both resting on the same floor (albeit in different locations). Another challenge when using automatically aligned fragments for training is that the learning procedure will likely be able to handle only a few false positive alignments before performance degrades substantially. This is also tricky because it is often hard to know when you have observed the same object twice (or two identical instances of the same object) versus two objects with similar appearance that are nevertheless distinguishable. There is a chicken-and-egg problem involved, because it is perhaps impossible for a learning-based system to know whether two objects are distinguishable until it has tried to learn to differentiate them.

8.1.3 Hierarchical Descriptor Spaces

It's not actually clear whether it is better to have a single, unified descriptor space to represent the appearance of multiple objects, as we had in Chapter 7, or whether it is better to use a hierarchical approach in which representations in one level of the hierarchy might be used to differentiate between classes of objects, another to differentiate instances of that class, and a lower level to learn an instance-specific descriptor space such as seen in Chapter 6 and related work by Florence, Manuelli, and Tedrake, (2018). We did not explore this in our experiments on the Jaech Gallery Dataset because we did not have access to any object labels or even segmentations, but if the merging strategy described above is successful in identifying multiple objects this might also be applicable in a self-supervised learning setting. This might be another way to improve convergence speed, as losses on the upper level of the hierarchy will look more like a classification loss between tens to hundreds of classes, which can be expected to lead to faster training compared to a contrastive loss over millions of vertices.

8.1.4 Online Learning

On the topic of convergence speed, it is a bit unsatisfying that all of the experiments in this document involved collecting large datasets and then learning from them offline. It would be interesting to see to what extent model-based self supervision can be used to learn descriptors online as observations come in. Even if the network needs a long period of offline training to get to a sufficient level of performance, it is certainly the case that a robot should continue to collect training data and fine-tune its perceptual capabilities, especially in situations where it is uncertain of its observations, and it would be desirable for this data to affect learning as soon as possible.

8.1.5 Active Learning

Another potentially promising and related direction is to move towards a more active learning procedure. In Chapter 7, we can see that the network still has trouble in the most extreme lighting settings such as direct sunlight. An actively perceiving and learning robot, however, could solve this by interposing itself or a manipulated object between light sources and surfaces and observing the changes in appearance caused by light and shadow that have no effect on scene geometry and thereby learn to ignore shadows when estimating surface correspondence. A robot could also move closer to obtain more observations of things that it does not fully recognize in order to gather more training data. Also, as mentioned in Chapter 7, a robot in a scene such as the Jaech Gallery could move objects around in order to segment them and then use the segmentation for training.

8.1.6 Learning from and Evaluating with Video

Another promising direction forward is to learn from video. While the datasets we collect are all based on video data, this is because the generative models that provide the labels operate on sequential data; we have not yet explored training networks that are given sequential input. This would allow the network to develop stronger representations over time and reduce uncertainty by integrating information from multiple observations. An interesting problem for fine-grained output with sequential input is that the network faces a data association problem as well — how to associate internal hidden representations of things it has observed in the past with current observations. Some promising work has been done in this direction by Xiang and Fox, (2017), and applying this or a similar technique to dense descriptor learning would be interesting.

8.1.7 Multimodal Self-supervised Learning

Adding in other sensor modalities will also be critical. There is a lot about the world that is much more easily learned with tactile and auditory sensors than with vision. A robot could use such sensors to record how surfaces 'feel' and what sounds they make when touched in various ways, and in an active perception scenario could intentionally collect such data to help disambiguate whether a current surface it is observing is or is not the same as a previously observed surface. Other sensors are worth exploring such as heat and pressure sensors, which are also used in human perception, but also some superhuman sensor capabilities such as sensors for magnetic and electric fields and hyperspectral imaging sensors.

Final Thoughts

Computer vision is a field that is currently in the midst of a revolution in which deep learning is rapidly advancing the state of the art on various tasks. This dissertation has put forward a few arguments about how these successes might be leveraged to also advance the state of the art in robotics by enabling embodied agents to visually perceive their local environments much more robustly, efficiently, and in greater detail. First, it was argued that embodied vision places different demands on vision systems that

require different design considerations. Another argument running through this dissertation is that generative models should not be discarded now that deep neural networks have arrived in full force. In many ways, generative models and learned discriminative techniques are complementary, and when used together the two approaches can each benefit from the other. It is likely that robots of the future will use generative models to reason about the world around them, but will also rely on learning methods to compress information from a lifetime of observations and allow for fast approximate inference. Perhaps they will use generative models to develop an understanding of the environments and events they observe, use this understanding as a basis for learning models of the world (such as appearance models, motion models, etc.), and use this learned knowledge to aid in efficiently maintaining an accurate understanding of the world. There is a lot more work to be done in this direction and it will be exciting to see it develop!

Bibliography

[1] Rareş Ambruş et al. "Meta-rooms: Building and maintaining long term spatial models in a dynamic world". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2014, pp. 1854–1861.

[2] Pedram Azad et al. "6-DoF model-based tracking of arbitrarily shaped 3D objects". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 5204–5209.

[3] Luca Ballan et al. "Motion capture of hands in action using discriminative salient points". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2012, pp. 640–653.

[4] Jan Bandouch, Odest Chadwicke Jenkins, and Michael Beetz. "A self-training approach for visual tracking and recognition of complex human activity patterns". In: *International Journal of Computer Vision (IJCV)* 99.2 (2012), pp. 166–189.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2006, pp. 404–417.

[6] Joao Bimbo et al. "Combining touch and vision for the estimation of an object's pose during manipulation". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2013, pp. 4021–4026.

[7] Jose-Luis Blanco. *A tutorial on se (3) transformation parameterizations and on-manifold optimization*. Tech. rep. 2013.

[8] Christoph Borst et al. "A humanoid upper body system for two-handed manipulation". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2007, pp. 2766–2767.

[9] Eric Brachmann et al. "Learning 6d object pose estimation using 3d object coordinates". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2014, pp. 536–551.

[10] Erik Bylow, Carl Olsson, and Fredrik Kahl. "Robust camera tracking by combining color and depth measurements". In: *International Conference on Pattern Recognition (ICPR)*. IEEE. 2014, pp. 4038–4043.

[11] Berk Calli et al. "Yale-CMU-Berkeley dataset for robotic manipulation research". In: *The International Journal of Robotics Research (IJRR)* 36.3 (2017), pp. 261–268.

[12] Daniel R Canelhas, Todor Stoyanov, and Achim J Lilienthal. "SDF tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2013, pp. 3671–3676.

[13] Maxime Chalon, Jens Reinecke, and Martin Pfanne. "Online in-hand object localization". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2013, pp. 2977–2984.

[14] Angel Chang et al. "Matterport3D: Learning from RGB-D Data in Indoor Environments". In: *International Conference on 3D Vision (3DV)*. IEEE. 2017, pp. 667–676.

[15] Lillian Chang, Joshua R Smith, and Dieter Fox. "Interactive singulation of objects from a pile". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 3875–3882.

[16] Will Chang and Matthias Zwicker. "Automatic registration for articulated shapes". In: *Computer Graphics Forum*. Vol. 27. 5. Wiley Online Library. 2008, pp. 1459–1468.

[17] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. IEEE. 2005, pp. 539–546.

[18] Christopher B Choy et al. "Universal correspondence network". In: *Neural Information Processing Systems (NIPS)*. 2016, pp. 2414–2422.

[19] Andrew I Comport, Éric Marchand, and François Chaumette. "Kinematic sets for real-time robust articulated object tracking". In: *Image and Vision Computing* 25.3 (2007), pp. 374–391.

[20] Mark Cummins and Paul Newman. "FAB-MAP: Probabilistic localization and mapping in the space of appearance". In: *The International Journal of Robotics Research (IJRR)* 27.6 (2008), pp. 647–665.

[21] Brian Curless and Marc Levoy. "A volumetric method for building complex models from range images". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM. 1996, pp. 303–312.

[22] Angela Dai et al. "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration". In: *ACM Transactions on Graphics (ToG)* 36.4 (2017), 76a.

[23] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2009, pp. 248–255.

[24] Guillaume Dewaele, Frédéric Devernay, and Radu Horaud. "Hand motion from 3d point trajectories and a smooth surface model". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2004, pp. 495–507.

[25] Tom Drummond and Roberto Cipolla. "Real-time tracking of highly articulated structures in the presence of noisy measurements". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. Vol. 2. IEEE. 2001, pp. 315–320.

[26] Tom Drummond and Roberto Cipolla. "Visual tracking and control using Lie algebras". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1999.

[27] Marius Fehr et al. "TSDF-based change detection for consistent long-term dense reconstruction and dynamic object discovery". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5237–5244.

[28] Pedro F Felzenszwalb and Daniel P Huttenlocher. "Distance transforms of sampled functions". In: *Theory of Computing* 8.1 (2012), pp. 415–428.

[29] Ross Finman et al. "Toward lifelong object segmentation from change detection in dense rgb-d maps". In: *Proceedings of the European Conference on Mobile Robots (ECMR)*. IEEE. 2013, pp. 178–185.

[30] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[31] Andrew W Fitzgibbon. "Robust registration of 2D and 3D point sets". In: *Image and Vision Computing* 21.13-14 (2003), pp. 1145–1153.

[32] Peter R Florence, Lucas Manuelli, and Russ Tedrake. "Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation". In: *Proceedings of the Conference on Robot Learning (CORL)*. 2018, pp. 373–385.

[33] Fadri Furrer et al. "Incremental Object Database: Building 3D Models from Multiple Partial Observations". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. 2018.

[34] David Gadot and Lior Wolf. "Patchbatch: a batch augmented loss for optical flow". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4236–4245.

[35] Varun Ganapathi et al. "Real-time human pose tracking from range data". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2012, pp. 738–751.

[36] Varun Ganapathi et al. "Real time motion capture using a single time-of-flight camera". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2010, pp. 755–762.

[37] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 580–587.

[38] John C Gower, Garmt B Dijksterhuis, et al. *Procrustes problems*. Vol. 30. Oxford University Press on Demand, 2004.

[39] Daniel Grest, Jan Woetzel, and Reinhard Koch. "Nonlinear body pose estimation from depth images". In: *Joint Pattern Recognition Symposium*. Springer. 2005, pp. 285–292.

[40] Raia Hadsell, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2006, pp. 1735–1742.

[41] Dirk Haehnel, Sebastian Thrun, and Wolfram Burgard. "An extension of the ICP algorithm for modeling nonrigid objects with mobile robots". In: *International Joint Conferences on Artificial Intelligence (IJCAI)*. Vol. 3. 2003, pp. 915–920.

[42] Steffen Haidacher and Gerd Hirzinger. "Estimating finger contact location and object pose from contact measurements in 3d grasping". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2. IEEE. 2003, pp. 1805–1810.

[43] Xufeng Han et al. "Matchnet: Unifying feature and metric learning for patch-based matching". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3279–3286.

[44] Bharath Hariharan et al. "Hypercolumns for object segmentation and fine-grained localization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 447–456.

[45] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[46] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2017, pp. 2980–2988.

[47] Thomas Helten et al. "Personalization and evaluation of a real-time depth-based full body tracker". In: *International Conference on 3D Vision (3DV)*. IEEE. 2013, pp. 279–286.

[48] Peter Henry et al. "Patch volumes: Segmentation-based consistent mapping with RGB-D cameras". In: *International Conference on 3D Vision (3DV)*. IEEE. 2013, pp. 398–405.

[49] Evan Herbst, Xiaofeng Ren, and Dieter Fox. "RGB-D object discovery via multi-scene analysis". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2011, pp. 4850–4856.

[50] Evan Herbst et al. "Toward object discovery and modeling via 3-d scene comparison". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 2623–2629.

[51] Katharina Hertkorn et al. "Virtual reality support for teleoperation using online grasp planning". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2013, pp. 2074–2074.

[52] Stefan Hinterstoisser et al. "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2011, pp. 858–865.

[53] Thomas Hulin et al. "The DLR bimanual haptic device with optimized workspace". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 3441–3442.

[54] Yangqing Jia et al. "Caffe: Convolutional architecture for fast feature embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 675–678.

[55] Andrej Karpathy, Stephen Miller, and Li Fei-Fei. "Object discovery in 3d scenes via shape analysis". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 2088–2095.

[56] Cem Keskin et al. “Real time hand pose estimation using depth sensors”. In: *Consumer depth cameras for computer vision*. Springer, 2013, pp. 119–137.

[57] Furkan Kirac, Yunus Emre Kara, and Lale Akarun. “Hierarchically constrained 3D hand pose estimation using regression forests from single frame depth data”. In: *Pattern Recognition Letters* 50 (2014), pp. 91–100.

[58] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the national academy of sciences* (2017), p. 201611835.

[59] Matthew Klingensmith et al. “Closed-loop Servoing using Real-time Markerless Arm Tracking”. In: *International Conference on Robotics And Automation (Humanoids Workshop)*. 2013.

[60] Michael C Koval et al. “Pose estimation for contact manipulation with manifold particle filters”. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2013, pp. 4541–4548.

[61] Michael Krainin et al. “Manipulator and object tracking for in-hand 3D object modeling”. In: *The International Journal of Robotics Research (IJRR)* 30.11 (2011), pp. 1311–1327.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Neural Information Processing Systems (NIPS)*. 2012, pp. 1097–1105.

[63] Nikolaos Kyriazis and Antonis Argyros. “Physically plausible 3d scene tracking: The single actor hypothesis”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 9–16.

[64] Hao Li, Robert W Sumner, and Mark Pauly. “Global correspondence optimization for non-rigid registration of depth scans”. In: *Computer Graphics Forum*. Vol. 27. 5. Wiley Online Library. 2008, pp. 1421–1430.

[65] Hao Li et al. “Realtime facial animation with on-the-fly correctives.” In: *ACM Transactions on Graphics (ToG)* 32.4 (2013), pp. 42–1.

[66] Yi Li et al. “DeepIM: Deep Iterative Matching for 6D Pose Estimation”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer International Publishing, 2018, 695–711.

[67] Hong Liu et al. “Multisensory five-finger dexterous hand: The DLR/HIT Hand II”. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2008, pp. 3692–3697.

[68] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440.

[69] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International Journal of Computer Vision (IJCV)* 60.2 (2004), pp. 91–110.

[70] Kendall Lowrey et al. “Physically-consistent sensor fusion in contact-rich behaviors”. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2014, pp. 1656–1662.

[71] Stephanie Lowry et al. “Visual place recognition: A survey”. In: *IEEE Transactions on Robotics* 32.1 (2016), pp. 1–19.

[72] Will Maddern et al. "1 year, 1000 km: The Oxford RobotCar dataset". In: *The International Journal of Robotics Research (IJRR)* 36.1 (2017), pp. 3–15.

[73] Michael J Milford and Gordon F Wyeth. "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 1643–1649.

[74] Antonio Morales et al. "Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2006, pp. 5663–5668.

[75] Raul Mur-Artal and Juan D Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras". In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262.

[76] Richard A Newcombe. "Dense Visual SLAM". PhD thesis. Imperial College London, June 2014.

[77] Richard A Newcombe, Dieter Fox, and Steven M Seitz. "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 343–352.

[78] Richard A Newcombe et al. "KinectFusion: Real-time dense surface mapping and tracking". In: *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2011, pp. 127–136.

[79] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. "Efficient model-based 3d tracking of hand articulations using kinect." In: *Proceedings of the British Machine Vision Conference (BMVC)*. Vol. 1. 2. 2011, p. 3.

[80] Karl Pauwels et al. "Real-time object pose recognition and tracking with an imprecisely calibrated moving RGB-D camera". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2014, pp. 2733–2740.

[81] Chen Qian et al. "Realtime and robust hand tracking from depth". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 1106–1113.

[82] Carl Yuheng Ren and Ian Reid. "A unified energy minimization framework for model fitting in depth". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2012, pp. 72–82.

[83] Jerome Revaud et al. "Deepmatching: Hierarchical deformable dense matching". In: *International Journal of Computer Vision (IJCV)* 120.3 (2016), pp. 300–323.

[84] Javier Romero et al. "Non-parametric hand pose estimation with object context". In: *Image and Vision Computing* 31.8 (2013), pp. 555–564.

[85] Renato F Salas-Moreno et al. "Slam++: Simultaneous localisation and mapping at the level of objects". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 1352–1359.

[86] Matthias Schröder et al. "Analysis of hand synergies for inverse kinematics hand tracking". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2013.

[87] John Schulman et al. "Tracking deformable objects with point clouds". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 1130–1137.

[88] Jamie Shotton et al. "Real-time human pose recognition in parts from single depth images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ieee. 2011, pp. 1297–1304.

[89] Jamie Shotton et al. "Scene coordinate regression forests for camera relocalization in RGB-D images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 2930–2937.

[90] Edgar Simo-Serra et al. "Discriminative learning of deep convolutional feature point descriptors". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2015, pp. 118–126.

[91] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Descriptor learning using convex optimisation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2012, pp. 243–256.

[92] Noah Snavely, Steven M Seitz, and Richard Szeliski. "Photo tourism: exploring photo collections in 3D". In: *ACM Transactions on Graphics (ToG)*. Vol. 25. 3. ACM. 2006, pp. 835–846.

[93] Srinath Sridhar, Antti Oulasvirta, and Christian Theobalt. "Interactive markerless articulated hand motion tracking using RGB and depth data". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2013, pp. 2456–2463.

[94] Jürgen Sturm et al. "CopyMe3D: Scanning and printing persons in 3D". In: *German Conference on Pattern Recognition*. Springer. 2013, pp. 405–414.

[95] Niko Sünderhauf et al. "Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free". In: *Proceedings of Robotics: Science and Systems (RSS)* (2015).

[96] Christian Szegedy et al. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[97] Jonathan Taylor et al. "The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2012, pp. 103–110.

[98] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. "Real-time seamless single shot 6D object pose prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).

[99] Jonathan Tompson et al. "Real-time continuous pose recovery of human hands using convolutional networks". In: *ACM Transactions on Graphics (ToG)* 33.5 (2014), p. 169.

[100] Markus Ulrich, Christian Wiedemann, and Carsten Steger. "CAD-based recognition of 3D objects in monocular images." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 9. 2009, pp. 1191–1198.

[101] Julien Valentin et al. “Exploiting uncertainty in regression forests for accurate camera relocalization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 4400–4408.

[102] Julien Valentin et al. “Learning to navigate the energy landscape”. In: *International Conference on 3D Vision (3DV)*. IEEE. 2016, pp. 323–332.

[103] Aaron Walsman et al. “Dynamic High Resolution Deformable Articulated Tracking”. In: *International Conference on 3D Vision (3DV)*. IEEE. 2017, pp. 38–47.

[104] Xiaolong Wang and Abhinav Gupta. “Unsupervised Learning of Visual Representations Using Videos”. In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE Computer Society. 2015, pp. 2794–2802.

[105] Lingyu Wei et al. “Dense human body correspondences using convolutional networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2016, pp. 1544–1553.

[106] Yandong Wen et al. “A discriminative feature learning approach for deep face recognition”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2016, pp. 499–515.

[107] Thomas Whelan et al. “ElasticFusion: Real-time dense SLAM and light source estimation”. In: *The International Journal of Robotics Research (IJRR)* 35.14 (2016), pp. 1697–1716.

[108] Thomas Whelan et al. “Kintinuous: Spatially Extended KinectFusion”. In: *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*. 2012.

[109] Simon AJ Winder and Matthew Brown. “Learning local image descriptors”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2007, pp. 1–8.

[110] Paul Wohlhart and Vincent Lepetit. “Learning descriptors for object recognition and 3d pose estimation”. In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2015, pp. 3109–3118.

[111] Yu Xiang and Dieter Fox. “DA-RNN: Semantic Mapping with Data Associated Recurrent Neural Networks”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2017.

[112] Yu Xiang et al. “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2018.

[113] Mao Ye and Ruigang Yang. “Real-time simultaneous pose and shape estimation for articulated objects using a single depth camera”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 2345–2352.

[114] Sergey Zagoruyko and Nikos Komodakis. “Learning to compare image patches via convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 4353–4361.

[115] Jure Zbontar and Yann LeCun. "Stereo matching by training a convolutional neural network to compare image patches". In: *Journal of Machine Learning Research* 17.1-32 (2016), p. 2.

[116] Andy Zeng et al. "3dmatch: Learning local geometric descriptors from rgb-d reconstructions". In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 199–208.

[117] Li Zhang and Jeffrey C Trinkle. "The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 3805–3812.

[118] Xu Zhang et al. "Learning Spread-Out Local Feature Descriptors." In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2017, pp. 4605–4613.

[119] Michael Zollhöfer et al. "Real-time non-rigid reconstruction using an RGB-D camera". In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 156.