TECHNISCHE UNIVERSITÄT DRESDEN

DOCTORAL THESIS

LEARNING SAMPLING-BASED 6D OBJECT POSE ESTIMATION

Author:
ALEXANDER KRULL
Born on 22 December 1982 in Aachen

Supervisor and First Referee:
Prof. PhD. ROTHER

Second Referee:
Prof. Ing. PhD. MATAS

Advisor:
Prof. Dr. GUMHOLD

Submitted:
20 November 2017

Defended:
16 January 2018

A thesis submitted in fulfillment of the requirements
for the degree of DOCTOR RERUM NATURALIUM (DR. RER. NAT.)

in the

CHAIR OF IMAGE PROCESSING
INSTITUTE OF ARTIFICIAL INTELLIGENCE

Declaration of Authorship

I hereby certify that I have authored this Dissertation entitled "Learning Sampling-Based 6D Object Pose Estimation" independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

ALEXANDER KRULL

"Nature laughs at the difficulties of integration."

–Pierre-Simon Laplace

Abstract

The task of 6D object pose estimation, *i.e.* of estimating an object's position (three degrees of freedom) and orientation (three degrees of freedom) from images is an essential building block of many modern applications, such as robotic grasping, autonomous driving, or augmented reality. Automatic pose estimation systems have to overcome a variety of visual ambiguities, including texture-less objects, clutter, and occlusion. Since many applications demand real time performance the efficient use of computational resources is an additional challenge.

In this thesis, we will take a probabilistic stance on trying to overcome said issues. We build on a highly successful automatic pose estimation framework based on predicting pixel-wise correspondences between the camera coordinate system and the local coordinate system of the object. These dense correspondences are used to generate a pool of hypotheses, which in turn serve as a starting point in a final search procedure. We will present three systems that each use probabilistic modeling and sampling to improve upon different aspects of the framework.

The goal of the first system, *System I*, is to enable pose tracking, *i.e.* estimating the pose of an object in a sequence of frames instead of a single image. By including information from previous frames tracking systems can resolve many visual ambiguities and reduce computation time. *System I* is a *particle filter* (PF) approach. The PF represents its *belief* about the pose in each frame by propagating a set of samples through time. Our system uses the process of hypothesis generation from the original framework as part of a proposal distribution that efficiently concentrates samples in the appropriate areas.

In *System II*, we focus on the problem of evaluating the quality of pose hypotheses. This task plays an essential role in the final search procedure of the original framework. We use a *convolutional neural network* (CNN) to assess the quality of an hypothesis by comparing rendered and observed images. To train the CNN we view it as part of an energy-based probability distribution in pose space. This probabilistic perspective allows us to train the system under the *maximum likelihood* paradigm. We use a sampling approach to approximate the required gradients. The resulting system for pose estimation yields superior results in particular for highly occluded objects.

In *System III*, we take the idea of machine learning a step further. Instead of learning to predict an hypothesis quality measure, to be used in a search procedure, we present a way of learning the search procedure itself. We train a *reinforcement learning* (RL) agent, termed *PoseAgent*, to steer the search process and make optimal use of a given computational budget. PoseAgent dynamically decides which hypothesis should be refined next, and which one should ultimately be output as the system's estimate. Since the search procedure includes discrete non-differentiable choices, training of the system via gradient descent is not easily possible. To solve the problem, we model PoseAgent's behavior as non-deterministic stochastic policy, which is ultimately governed by a CNN. This allows us to use a sampling-based *stochastic policy gradient* training procedure.

We believe that some of the ideas developed in this thesis, such as the sampling-driven probabilistically motivated training of a CNN for the comparison of images or the search procedure implemented by PoseAgent have the potential to be applied in fields beyond pose estimation as well.

Acknowledgements

First of all, I would like to thank my supervisors. I want to thank Carsten Rother for his introduction and guidance in the world of computer vision research, but also for creating the wonderful work environment that was his research group in Dresden. I want to thank Stefan Gumhold for his much appreciated feedback and the discussions we had, especially on the problems of differential geometry. I also want to thank my previous supervisors Iva Tolić and Uwe Petersohn, who guided me during my first publication and played an important role in my decision to be a PhD student.

In the last year, I was able to spend some time as a visitor at Microsoft Research in Cambridge. I want to thank Sebastian Nowozin and Jamie Shotton, but also again Carsten Rother, for making this awesome and intellectually simulating experience possible. I sincerely enjoyed Sebastian's and Jamie's supervision and collaboration during this time.

I owe gratitude to many of the people I met at during my time as a student in Dresden. My fellow PhD students and office mates Frank Michel and Eric Brachmann contributed a lot in making my research possible. I am thankful for all their input, discussions, and the good times we had at the office. Remembering my time as a diploma student, I am thankful to Dimitrij Schlesinger and Boris Flach who helped to spark my interest in probabilistic methods. I also have to thank Joachim Staib, for his much appreciated help with the problems of software rendering, Linux, and the CGV software framework.

Finally, I want to express the deepest gratitude for the help, support, and love I have received from my family and friends. I want to thank my girlfriend Dorit for all her patience and encouragement through busy times and also for her help in proof-reading. I want to thank my dear friend Peter for reading part of the thesis and for being a true friend. I want to thank Eva and Achim for their sympathy and advice. I want to thank my brother Cornelius for his comments, discussion, and counsel, as well as for being a brother in the best sense imaginable. Last but certainly not least, I want to thank my mother and father for their continuous and unconditional support.

List of Publications

During my thesis I present the methods from the following previously published papers:

- *PoseAgent: Budget-Constrained 6D Object Pose Estimation via Reinforcement Learning*
  Alexander Krull, Eric Brachmann, Sebastian Nowozin, Frank Michel, Jamie Shotton, Carsten Rother
  CVPR 2017

- *Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images*
  Alexander Krull, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, Carsten Rother
  ICCV 2015

- *6-DOF Model-Based Tracking via Object Coordinate Regression*
  Alexander Krull, Frank Michel, Eric Brachmann, Stefan Gumhold, Stephan Ihrke, Carsten Rother
  ACCV 2014

I additionally contributed to the following papers on similar topics during the work on the thesis. They are however not explicitly discussed in the thesis.

- *DSAC - Differentiable RANSAC for Camera Localization*
  Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, Carsten Rother
  CVPR 2017

- *Global Hypothesis Generation for 6D Object Pose Estimation*
  Frank Michel, Alexander Kirillov, Eric Brachmann, Alexander Krull, Stefan Gumhold, Bogdan Savchynskyy, Carsten Rother
  CVPR 2017

- *Random Forests versus Neural Networks - What's Best for Camera Relocalization?*
  Daniela Massiceti, Alexander Krull, Eric Brachmann, Carsten Rother, Philip H.S. Torr
  ICRA 2017

- *Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image*
  Eric Brachmann, Frank Michel, Alexander Krull, Michael Ying Yang, Stefan Gumhold, Carsten Rother
  CVPR 2016

- *Pose Estimation of Kinematic Chain Instances via Object Coordinate Regression*
  Frank Michel, Alexander Krull, Eric Brachmann, Michael Ying Yang, Stefan Gumhold, Carsten Rother
  BMVC 2015

- *Learning 6D Object Pose Estimation using 3D Object Coordinates* Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, Carsten Rother
  ECCV 2014

Contents

Chapter 1

Introduction

Contents

The idea of solving difficult numerical problems through computerized random sampling goes back roughly 70 years. Stanislaw Ulam, Nicholas Metropolis, and John von Neumann first applied the technique during their work in the Manhattan project [1]. In their 1949 paper [2] Metropolis and Ulam already argued for application of this idea

for problems as diverse as the solving integrals in high dimensional space, inference in probabilistic models, and studying the penetration of cosmic rays in the atmosphere. The technique has since become known as the *Monte Carlo* method. The development of the *Metropolis algorithm* [3] in 1953, which allowed Monte Carlo sampling from difficult distributions lead to an enormous success of the approach.

Nowadays, the Monte Carlo method is applied in a variety of fields [4] such as astronomy [5], biology [6], chemistry [7], geology [8], electrical engineering [9], medicine [10], finance [11], or even archeology [12]. Likewise, it is widely used in the field of artificial intelligence and computer vision. Here, it has become an essential tool for taming complex probabilistic models [13], allowing us to perform inference, where it would otherwise be infeasible.

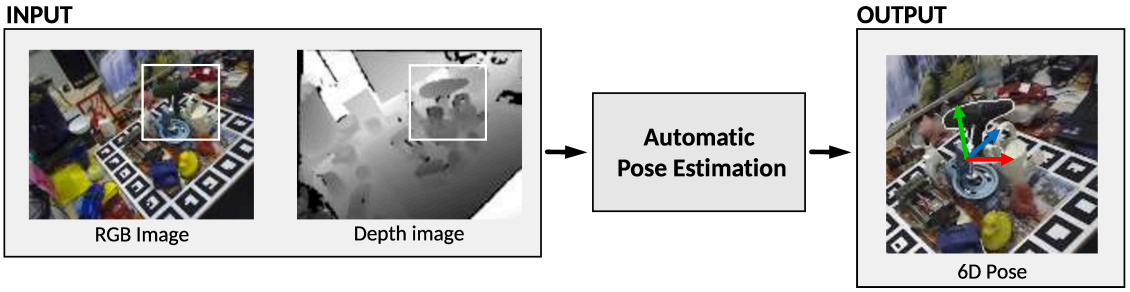


Figure includes images and models from the dataset by Hinterstoisser *et al.* [14], published in 2012 under a CC BY 4.0 Licence.

FIGURE 1.1: The 6D object pose estimation task: Given an RGB-D image and an object of interest, we want to estimate the position and orientation (the *pose*) of the object.

During this thesis we will examine the computer vision problem of *6D object pose estimation* (Figure 1.1), the task of finding an object's position and orientation (the *pose*) from a visual input.[1] Any automatic system trying to solve this task has to address a multitude of uncertainties and visual ambiguities, arising from clutter, occlusion, or similar problems. In this thesis we will demonstrate that probabilistic models can help to cope with these uncertainties and that the Monte Carlo method is the right tool for their implementation.

To appreciate the potential impact of automatic pose estimation systems, it is worth considering the importance of the human brain's ability in this regard. Pose estimation is deeply tied to our skill in grasping objects with our hands. This process of manual grasping and manipulation is known as *prehension*. While humans are generally capable of blind prehension, we usually rely on visual information to first assess the pose of the object we want to grasp [15, 16]. In fact, it is a widely accepted hypothesis in neuroscience that the human brain contains two separate visual systems [17–19]. While one, termed *ventral system*, provides us with information to construct a

[1] The pose of an object has six degrees of freedom, three for the position and three for the orientation. we will use the term *object pose estimation* synonymously.

conscious model of our environment, the second *dorsal system* is profoundly connected to our motor skills at an non-conscious level. When we attempt to grasp an object, it is the dorsal system that provides the assessments of the object's position, orientation, and shape required for a skillful grasp.

Although humans are not the only animals with the ability for dexterous prehension, the role it played for human cultural and technological development can hardly be overestimated [20]. This ability is an essential requirement for handling basic tools, be it a stone, a handaxe, or a screwdriver. It enabled us to arrange rocks, timber, and bricks into buildings, and later allowed us to produce the goods we consume and to assemble the machinery we rely on to sustain our life style.

Nowadays, many production processes have been automated and tasks previously done by the human hand and visual systems are now executed by robots. For the most part however, these robots operate in the controlled environment of a factory and perform only highly specialized and repetitive tasks that do not require a much flexibility. As a result the human brain's ability for prehension and pose estimation is still essential to the economic processes of our world. For instance, the manufacturing of products or the construction of buildings still relies on our manual skills and the use of basic hand held tools. Beyond this, very basic and even mindless tasks are also still performed by humans: For example, workers are still required to manually pick items from a shelf to place them in the delivery packages of the online retailers and supermarkets [21, 22]. Sometimes they work in warehouses where almost all other processes are completely automated and designed around the human ability of prehension, which robotic systems don not yet posses reliably enough.

Prehension is of course not the only human ability relying on our brain's capacity for pose estimation. We require pose estimation on a larger scale whenever we drive a car, ride a bicycle through traffic, or when we navigate as a pedestrian across a busy street. In such a situation our brain has to estimate the position and orientation of vehicles. It furthermore has to assess their movement and speed, in real-time to allow a safe participation in traffic [23, 24].

In order to allow robotic systems to carry out tasks involving prehension or navigation in traffic, we need to equip them with a capacity for pose estimation comparable to our own. Automatic pose estimation becomes a key building block in enabling robotic systems to interact with our complex material world. Robotic systems require this skill to execute mindless tasks as the filling of packages for delivery [25], but also to collaborate with us by handing us objects or receiving them from us [26]. Autonomous vehicles are already at the verge of revolutionizing the way we travel and transport goods. They navigate busy traffic with pose estimation playing an important role [27–29].

Another important application for pose estimation comes from a different realm, summarized under the term *augmented reality* (AR). AR is the real-time composition of real and computer generated images to create the illusion of a modified reality.AR systems such as *HoloLens* [30] can be used for different purposes: They can combine our visual reality with elements of a computer game for entertainment, or annotate real world objects with additional useful information in a professional setting. In order to create a believable AR experience, the virtual content must be rendered under the correct perspective. Thus, to modify the appearance of a real object, its pose relative to the viewer has to be estimated first.

Current automatic pose estimation systems function well in some settings. They however still have to overcome a number of challenges before approaching the human level of robustness and reliability. Automatic pose estimation most importantly has to address said visual ambiguities such as clutter or occlusion. Considering that many applications require real-time performance an automatic pose estimation system has to additionally cope with its potentially limited computational resources.

We believe that probabilistic approaches can be the next step towards resolving these challenges. In this thesis, we will use probabilistic models to address the problems of visual ambiguity and the distribution of computational resources. We will use machine learning to train said models and rely on Monte Carlo methods to make training and inference feasible.

Our methods will build upon on the multi step pose estimation framework introduced by Brachmann *et al.* [31]. This framework will serve as our starting point for exploring different sampling-based approaches. Sampling will allow us to cope with the uncertainties and visual ambiguities of the task, and it will help us to train a system to make optimal use of a computational budget. We believe that the techniques we develop in this thesis are not restricted to the problem of pose estimation, but can be useful in other areas as well.

We will present these techniques in three systems, each focusing on different components of the original framework and each addressing different challenges of the pose estimation problem.

The first system (*System I*) we will present, is an adaption of the framework from [31], which processes one image at a time, for real-time pose tracking, *i.e.* for processing a sequence of images. Tracking methods have a potential advantage over methods that process single images as they can make assumptions about the motion of an object between frames and resolve ambiguities over time. We use a *particle filter* (PF) [32] approach that represents uncertainty in every video frame as a set of samples, propagated through time.

In our second system (*System II*), we will focus on the problem of assessing the quality of pose hypotheses. This is especially challenging when the object is strongly occluded. We propose a *convolutional neural network* (CNN) that compares rendered and observed images to solve this task. To train the CNN, we use a probabilistically motivated sampling procedure.

Finally, in our third system (*System III*) we will replace a static search procedure in [31] with a dynamic *reinforcement learning* (RL) [33] -based system that is trained to optimally distribute a limited computational budget. Training of this system becomes possible through a sampling-based approach.

We will now continue this chapter by describing the different variants of the pose estimation task in Section 1.1. In the following Section 1.2 we will discuss the most important challenges for a pose estimation system. In Section 1.3 we will provide a survey of the related work on the topic. In Section 1.4 we will give an brief overview of the three systems we will present in this thesis. Finally, We will conclude the chapter by summarizing our main contributions, in Section 1.5, and outlining the structure of the remaining thesis in Section 1.6.

1.1 Different Tasks of 6D Object Pose Estimation

In literature the term pose estimation is used to a refer to variety of different tasks. In the course of this thesis we will focus on two specific formulations, the tasks of *one-shot pose estimation* and online *pose tracking*.

One-shot pose estimation is the problem of finding an object's position and orientation in a single image. Pose tracking on the other hand is the task of estimating position and orientation of an object in an image sequence. In the following sections we will start by providing a definition for these tasks (Sections 1.1.1 and 1.1.2). We will then go on to discuss the related problems of *object detection* (Section 1.1.3) and *camera pose estimation* (Section 1.1.4).

1.1.1 One-Shot Pose Estimation

One-shot pose estimation describes the problem of estimating an object's pose from a single image, as opposed to a sequence. Many methods, e.g. [14, 31, 34, 35], focus on this problem, as it is the foundation for many other tasks.

In this thesis we will use the term to refer to the following specific problem: Given an input RGB-D input image $\boldsymbol{x}$, our goal is find an estimate $\hat{H}$ of the pose of a particular object c relative to the camera. We make the following assumptions:

- Exactly one instance of the object is present in the image.

- A 3D model of the object is known.
- All internal camera parameters are known.

We consider our image $\boldsymbol{x} = (I, \boldsymbol{d})$ to consist of an RGB component $I = (I_1 \dots I_n)$ and a depth component $\boldsymbol{d} = (d_1 \dots d_n)$. With n denoting the number of pixels in the image.

The terms I_i and d_i shall denote the color and depth information for a particular pixel i. We will use $\boldsymbol{d}_i$ when referring to the 3D coordinates of the pixel in the coordinate system of the camera, as opposed to d_i when referring to the 1D depth vale information. The 1D depth value d_i is equal to the negative z-coordinate of the camera coordinates $\boldsymbol{d}_i$. Since, all internal camera parameters are known, we can calculate $\boldsymbol{d}_i$ from d_i.

During this thesis, we will often rely on additional features computed on $\boldsymbol{x}$ as a type of preprocessing. To allow a simplified notation, we summarize the original image $\boldsymbol{x}$ and such precomputed features as *observation*, denoted by $\boldsymbol{z}$.

Let us now consider the pose $H = (R, \boldsymbol{v})$. It consists of a rotational component, in matrix description [2] R and a translational component described as a 3D vector $\boldsymbol{v}$. Together, they describe a transformation that maps a point $\boldsymbol{y}_i$ from the local coordinate system of the object to its representation in the coordinate system of the camera. This transformation is defined as the application of the rotation followed by the application of the translation. It will be denoted as

$$H(\boldsymbol{y}_i) = \boldsymbol{v} + R\boldsymbol{y}_i. \tag{1.1}$$

Such transformations are known as *rigid body transformations*. They will receive a more thorough discussion in Chapter 3.

1.1.2 Pose Tracking

We can define the pose estimation problem for a sequence of images, as opposed to a single image. This kind of task will be referred to as pose tracking.

In a situation where a sequence of images is available as input, tracking an object can have a number of advantages compared to doing repeated one-shot pose estimation for each image separately. By making assumptions about the movement of an object, a tracking system can help to resolve many ambiguities. Even when an object is fully occluded, a tracking system can still provide a sensible estimation for the object's pose, by assuming that the object moves smoothly and with a limited speed. Tracking problems can be formulated as an *offline* or as an *online* task.

[2] We will give a discussion of rotation matrices in Section 3.1.1

In the offline formulation of pose tracking we are given a sequence of observations $\boldsymbol{z}_1 \dots \boldsymbol{z}_T$ and our task is to find estimates $\hat{H}_1 \dots \hat{H}_T$ of the corresponding poses. Making assumptions about the motion of the objects in between frames can help us to find a joint estimate, which is superior to separate estimates done frame by frame.

In this thesis we will focus on the online formulation of pose tracking (see e.g. [36–39]). Here, we are not given the entire sequence of observations, but only observations up the current point in time t. Our task is now to find an estimate $\hat{H}_t$ of the current pose while making use of all previous observations $\boldsymbol{z}_1 \dots \boldsymbol{z}_t$.

This kind of formulation is used in systems, where new observations come in sequentially and have to be processed in real-time to create an updated estimate for every time step. Consider for example an augmented reality application adjusting the perspective of virtual elements to react to the movements of a real object, or a robot attempting to grasp an object handed to it by a human.

1.1.3 Pose Estimation and Object Detection

The term object detection often refers to the problem of finding the 2D bounding boxes that contain different objects, e.g. [40, 41]. Many methods in literature combine the tasks of object pose estimation and object detection. They essentially drop the assumption we made in Section 1.1.1 that the object is present exactly once. Such a task is studied e.g. in [14, 36, 38, 42].

While we see this task as highly practically relevant, we will not consider this problem during this thesis. Instead, we will focus on the task from Section 1.1.1 and the online tracking task from Section 1.1.2.

1.1.4 Camera Pose Estimation

Camera pose estimation is the task of finding the position and orientation of the camera from an observed image taken in a known scene (see e.g. [43–46]). The problem is structurally identical to the problem of object pose estimation.

In both tasks–object pose estimation and camera pose estimation–we want to find the rigid body transformation between two coordinate systems. To make the equivalence clear, we can view the entire environment as the object we are interested in.

In practice there are however some differences to be considered. The most important one in our opinion is the following: In camera pose estimation, every pixel in the image belongs to the object, *i.e.* the scene, and potentially contains information about the pose we are interested in. On the other hand, in object pose estimation often only a small subset of pixels is part of the object while the vast majority will belong to an uninformative background.

Camera pose estimation will not play an explicit role in this thesis.

1.2 Challenges

a) b) c)

FIGURE 1.2: Visual ambiguities in pose estimation: **a):** Different poses can have a similar appearance. Top: Texture-less objects make it difficult to distinguish between different poses. Bottom: Some objects appear symmetric when viewed from a certain perspective. This can make it impossible to estimate the correct pose. **b):** An object can have variable appearance even under the same pose. Top: Lighting conditions can drastically change the appearance of an object. Bottom: Reflective and transparent materials can change the object's appearance depending on background, and perspective. **c):** Sometimes a combination of a) and b) can occur. Top: A cluttered background can have a highly variable appearance, but can also contain elements that look similar to the object of interest, making it difficult to distinguish between different possible positions. Bottom: Occlusion changes the appearance of an object, but it also reduces available information and can make it hard to distinguish different poses.

An automatic pose estimation system has to face a number of different challenges when it is used in a non-controlled real life environment. Here, we want to attempt a categorization. Most challenges have to do with visual ambiguity of some kind. We want to distinguish the following three categories of visual ambiguity:

- Some objects have a *similar or identical appearance* when viewed under different poses. Texture-less objects and symmetric or seemingly symmetric objects fall into this category.
- Sometimes a single object can have a highly *variable appearance* even when viewed from the same perspective. We see lighting as well as transparent or reflective materials as factors that can make the appearance of an object highly variable.
- Sometimes both said issues occur jointly and have a common origin. We will argue that clutter and occlusion are such *combined challenges*.

Finally, we will discuss a challenge that we see as critical, but that does not arise from visual ambiguity, namely the *efficient use of computational resources*.

1.2.1 Similar Appearance of Different Poses

Texture-less Objects

Texture-less objects are challenging (see Figure 1.2 a, top). It is difficult to distinguish between the different perspectives of an object without texture.

Especially many traditional methods for pose estimation rely on detecting key-points-based on texture [14]. Texture-less objects are however quite common in real life scenes. The systems we will present in this thesis do not require the detection of interest points, and are thus not as susceptible to the issue.

Recently Hodaň *et al.* [47] published a benchmark dataset specialized on the problem of texture-less objects. We have not yet evaluated the methods presented in this thesis on this dataset. However, the datasets [14, 31, 48] we use in this thesis include many objects without, or with very little texture.

Symmetric Objects

Objects can be symmetric in a variety of different ways. Some objects are continuously rotationally symmetric around one axis. Consider for example a plate or a vase. Other objects exhibit more complex discrete symmetries, such as a cube (see Figure 1.3). For some objects these symmetries are nearly perfect, for other objects there are small hints that break the symmetry. For the former it is impossible to determine the precise 6D pose, because there are multiple correct answers. For the latter it is very challenging. The dataset from [47] includes a variety of such symmetries and self similarities.

Sometimes objects only appear be symmetric when viewed from a certain perspective. Consider for example a cylindrical coffee mug viewed from a perspective that is hiding its handle (see Figure 1.2 a, bottom). When viewed from such a perspective it is impossible to pinpoint a single correct 6D pose, as long as the handle is not visible.

Recently Hodaň *et al.* [49] argued that one should consider this fact when measuring the error of a pose estimation system. They suggest to use a novel error measure, termed *visual surface discrepancy* to punish only visible errors.

1.2.2 Variable Appearance under a Fixed Pose

Lighting

The same object can appear radically different under different illumination (see Figure 1.2 b, top). Illumination can include arbitrarily complex patterns and is difficult to estimate. As a result many systems try to deal with the problem by using features

Image by Zumthie, published 2009 on German Wikipedia, public domain.

FIGURE 1.3: The Platonic solids exhibit many discrete symmetries.

that are invariant to light conditions, e.g. [14, 31]. A system that can rely on depth sensor has a big advantage in this respect, as depth sensors can produce a mostly illumination invariant image of the object. In [31] Brachmann *et al.* present a dataset specialized on difficult light conditions. It is however quite unrealistic with respect to background variability or clutter. We will not focus on difficult lighting in this thesis.

Difficult Materials

Certain materials can make pose estimation challenging (see Figure 1.2 b, bottom) [50]. Apart from few exceptions, e.g. [51, 52], the vast majority of pose estimation methods has focused on objects made from opaque and approximately Lambertian materials. The appearance of such materials is largely independent of the perspective and their surroundings. Reflective and transparent materials however, can change their appearance drastically depending on their surroundings and the relative position of the camera. To complicate things further, depth sensors are often not able to provide reliable estimates for such materials.

In this thesis, we will not explicitly examine the problem of difficult materials. We see it however as an important problem for future research.

1.2.3 Combined Challenges

Cluttered Background

Scenes with a large amount of clutter are challenging (see Figure 1.2 c, top) [14]. It can be argued that the problem of clutter creates both kinds of ambiguities we mentioned.

On the one hand, different poses of an object in a cluttered scene may have a similar appearance: If the scene contains an object that is visually similar to the object we are interested in, a pose estimation system might confuse the positions of the two. The appearance would be similar if the two objects were switched.

On the other hand, a cluttered scene can lead to a highly variable appearance of an object under the same pose: Since the clutter itself is highly variable the same object under the same pose can result in highly variable images.

The datasets [14, 31, 48] we will use during this thesis reflect this challenge well.

Occlusion

The potential occlusion of objects (see Figure 1.2 c, bottom) is especially challenging [31]. It also introduces both kinds of ambiguity.

On the one hand, different poses of an occluded object will have a similar appearance, whenever important features of the object are hidden. The more of an object is hidden the more similar will be its appearance under different poses. Considering the extreme case, if an object is fully occluded, all different orientations of the object will produce identical images.

On the other hand, occlusions can occur in a countless number of ways. The same object under the same pose can appear very differently depending on which part of the object is occluded and what is the appearance of the occluding object.

For some depth sensors occlusions will cause a depth shadow artifact around the borders of an occluding object. This can make pose estimation even more challenging.

In general, methods which make use of local information, such as local feature descriptors or predicted correspondences are at an advantage here, compared to methods that rely on the appearance of the object in its entirety.

Occlusion will be one of the challenges we focus on. We will evaluate our methods on datasets [14, 31, 48] featuring heavy occlusion.

1.2.4 Efficient Use of Computational Resources

Finally, we would like to mention the challenge of making efficient use of limited computational resources. In pose estimation as in many other problems, there is a trade-off between computation time and the quality of the results. Spending more computation time to find the pose of an object in a cluttered scene will increase the chance of finding it. In situations however, where computation time is limited or costly, the designer of a pose estimation system has to account for this. This is especially true for real-time tracking systems [53]. A system that spends to much time computing the pose in a particular frame, might miss the next frame coming from the camera and potentially loose the object altogether if it is moving quickly. In other words inefficient use of the computational resources at one point in time, can lead to the reduced accuracy in later time points.

1.3 Related Work

In this section we will provide a general overview of approaches for the problem of 6D pose estimation and camera pose estimation. At this point, we do not include

the work that is methodologically related to the specific systems presented in Chapters 4, 5,and 6. Such overviews can be found in Sections 4.2, 5.2, and 6.2, respectively.

A variety of different approaches can be found in literature. We divide the methods into five major categories:

- *Sparse key-point-based methods* first detect significant key-points in the image. They compute descriptors for the region around these key-points and match them to previously determined points on the surface of the object.
- *Template-based methods* create a set of appearance templates for different discrete perspectives. The templates are then compared with different parts of the image, usually in a sliding window scheme.
- *Voting methods* aggregate local information by allowing pixels or image patches to vote for pose hypotheses.
- *Dense correspondence-based methods* use a trained discriminative function to predict pixel-wise correspondences between the coordinate system of the object and the coordinate system of the camera. The information of multiple pixels is then combined to create pose hypotheses. This is the approach we will follow.
- *Direct regression methods* use machine learning to directly predict the pose from an observed image.

1.3.1 Sparse Key-Point-Based Methods

Such methods consist of two major steps: First, the system detects a set of key-points in the image and calculates descriptors for their local neighborhoods. Then, these descriptors are matched with descriptors on sparse 3D model, which has been constructed beforehand. Often a RANSAC [54] search is used to find sets of correspondences to produce pose hypotheses.

The so called SIFT (scale-invariant feature transform) features, introduced by Lowe in [55] provide arguably the most widely known algorithm to detect and describe such local key-points. Following a similar method by Rothganger *et al.* [56], Gordon and Lowe [57] describe a sparse key-point-based method for object pose estimation using SIFT features.

The basic approach was the foundation of multiple more efficient systems such as [58] or [59].

A major advantage of sparse key-point-based methods is their robustness to clutter as well as to partial occlusion. The major drawback of the approach is its reliance on the texture of the object. When dealing with texture-less objects, key-points can often only be found along the edge of an object. This is highly problematic because such points can drastically change their appearance, when the object is slightly rotated.

1.3.2 Template-Based Methods

Template-based methods can be a solution when dealing with texture-less objects. Such methods store descriptors, called templates, for the appearance of the entire object under different orientations. Even if the object is texture-less the shape of the object's silhouette as a whole can be informative enough to identify a template. Often, these methods use a sliding window approach to compare their templates to different image locations. If a well fitting template is found the accordant orientation can be used together with the location in the image to give an estimate of the pose.

A downside of this approach is that the exhaustive sliding window search can be computationally expensive. However, a number of solutions has been suggested to address this issue. A second more severe issue is that partially occluded objects often fail to be matched with their template.

Sliding window template matching has been used for a long time to localize 2D patterns in images [60]. In such methods the template was often a small image patch and the system would calculated simple distance measures as between the pixel intensities in the image and in a template.

More recently Hinterstoisser *et al.* [14] presented an extremely successful template matching system for 6D pose estimation. Hinterstoisser *et al.* create templates by rendering the object under different perspectives. Instead of storing images for each template they calculate gradient-based discretized descriptors, which allow them to do extremely efficient comparisons. While their sliding window matching algorithm is in principle linear in complexity with the number of templates, Hinterstoisser *et al.* achieved impressive speeds by using a highly efficient implementation.

More recently Kehl *et al.* [61] build on the system from [14] and achieve a sublinear complexity in the number of templates by using a hash function to reduce the number of required comparisons. In [62] Konishi *et al.* use a hierarchical tree system to improve the scalability of the matching procedure.

Other authors focus on developing learned template descriptors for pose estimation. In [35] Rios-Cabrera and Tuytelaars use the general pipeline from [14] but train their templates in a discriminative way to increase accuracy. A modified cascaded scheme for template matching allows them to greatly speed up their system. In [63] Wohlhart and Lepetit use a CNN to map image patches to a descriptor space, where they perform a nearest neighbor search among templates to find an approximate pose. The system is trained to ensure that images of similar pose are mapped to similar positions in the pose space. Once the CNN is trained it can be used with unseen objects as well.

Note that some methods combine the idea of templates with a voting scheme. We will discuss them in Section 1.3.3.

All in all, template-based methods have achieved impressive results and computation times. They deal well with texture-less or symmetric objects and cluttered backgrounds, but still struggle with occlusion.

1.3.3 Voting Methods

Voting methods allow image regions or even individual pixels to cast votes for plausible pose estimates. Similarly to key-point-based methods, voting methods aggregate information from multiple local sources. This makes them inherently more stable with respect to occlusion.

The idea of voting goes back to *Hough voting schemes* [64] introduced by Duda and Hart to estimate the position and orientation of 2D lines. In their scheme every pixel casts votes in a discretized 2D parameter space. The local maxima in this space are later used to identify lines. In [65] Ballard showed how to generalize the idea to arbitrary shapes.

More recently Sun *et al.* [66] and Gall *et al.* [67] applied the same general approach for the detection and coarse pose estimation of real world objects.

In [34] Tejani *et al.* use Hough voting for 6D pose estimation. They use a *random forest* [68] to process the image and cast votes for the position of the object. The rotational component is found in a second step.

Doumanoglou *et al.* [69] train an autoencoder system to map sampled image patches to a descriptor space. In a second step, they use a random forest, which ultimately casts votes directly into the 6D pose space.

Some modern methods combine the idea of voting in combination with templates (see Section 1.3.2). In such methods, a patch usually directly votes for a template-based on its similarity. The multi stage pipeline presented by Hodaň *et al.* in [42] uses a voting scheme to perform template-matching-based on the template construction from [14]. The method achieves sublinear complexity.

Another example is [70] by Kehl *et al.*. Like [69], they use an autoencoder method to learn a mapping from image patches to a descriptor space. During test time they sample patches from the image and allow them to distribute votes based on the similarity to a set of templates.

Drost *et al.* [71] use voting in a geometrically driven approach to find the pose of an object from a 3D point cloud. They select reference points and pair each of them with all other points. Then, they calculate features on these point pairs and use them in a voting scheme. Votes for each reference point are cast in its own discrete voting array, which is only 2D. The voting scheme accomplishes two things: (i) It finds a correspondence between the reference point and a point on the surface of the 3D Model of the object, this coincides with one dimension of the array. (ii) It finds

the 1D rotation angle of the point around its normal. When combined, these two parameters allow the direct calculation of a pose hypothesis. In [71] this hypothesis generation is followed by a clustering operation in pose space.

Hinterstoisser *et al.* [72] build on this system and increase its performance and speed by streamlining the pairing and voting process.

Voting methods are able to cope well with occlusion and most other said challenges. We see voting as a promising approach for future systems.

1.3.4 Dense Correspondence-Based Methods

The systems we will present in this thesis all fall into the category of dense correspondence-based methods. Such methods use a trained function to predict correspondences between the pixels of the image and positions in the local coordinate system of the object. Similarly to the key-point-based methods, multiple correspondences are selected in a RANSAC like fashion and combined to generate pose hypotheses.

The general idea of densely predicting correspondences was first introduced for the problem human body pose estimation. In the *Virtuvian Manifold* [73] Taylor *et al.* address this problem by predicting each pixel's position on the surface of the human body, before finally fitting a parametric model.

Following this work, Shotton *et al.* describe a similar system for camera pose estimation in known scenes. They predict each pixel's position in the coordinate system of the scene and use it as stepping stone for their final estimate.

Inspired by this approach, Brachmann *et al.* [31] published a method for 6D object pose estimation. This method will provide the framework for the systems we will present in this thesis. We will give a description of the framework in Chapter 2. An overview is provided in Figure 1.4.

The method from [31] uses a random forest to jointly predict whether or not a pixel is part of the object and where it is located in the object's local coordinate system, in case it is part of the object. The former prediction is referred to as *object probabilities*, the latter prediction as *object coordinates*.

Brachmann *et al.* use a RANSAC-based sampling scheme to generate a pool of pose hypotheses, which serve as starting point for a search procedure. During this procedure, pose hypotheses are evaluated via a scoring function, based on analysis-by-synthesis: The function uses a 3D model of the object to render images under the pose hypothesis and then does a pixel-wise comparison against observed images.

The framework from [31] unifies a number of advantages: (i) The predicted correspondences can be trained to be invariant against lighting changes. (ii) By using

densely predicted correspondences, the method is able to combine some inherent robustness against occlusion, as found in sparse key-point-based methods, with an ability to handle texture-less objects. (iii) They perform well in cluttered scenes, as their hypothesis-based search and analysis-by-synthesis-based scoring allows them to effectively tell similar looking objects apart. The systems presented in this thesis will build on [31] to benefit from these advantages.

We see the biggest limitation of [31] in its handling of occluded objects. Even though dense predictions bring inherent robustness to occlusion, the simple analysis-by-synthesis approach struggles, when applied with occluded objects, which can have a dramatically different appearance compared to an unoccluded rendered 3D model. In *System II* we will present a machine learning-based way to improve this scoring function.

While Brachmann *et al.* also use sampling to generate hypotheses, their view on sampling differs significantly from the one we will take in this thesis. The RANSAC-based sampling procedure in [31] functions as a black box, producing hypotheses. The systems, we will present in this thesis will include this procedure as well. However, they will additionally introduce different Monte Carlo sampling schemes in the various parts of the framework.

In Contrast to [31], we will always view sampling as a way to represent various probability distributions we are interested in. In *Systems I* and *II* sampling is used to represent a posterior distribution over poses that ultimately coincides with our knowledge of the pose. In *Systems III* we use samples to describe a probability distribution, governing the behavior of an RL agent.

Apart from the systems to be presented in this thesis, multiple other works have successfully build upon [31]. In [74] Michel *et al.* extend the method for objects, which are not rigid, but include moving parts, such laptop computers or cupboards with drawers and doors. In [75] Mund *et al.* adapt the system for the use with a *LiDAR* sensor to improve airfield safety. In [45] the system is modified to work without depth information and applied to camera pose estimation. Finally, in [46] Brachmann *et al.* replace the random forest with a CNN and find an RL inspired perspective to allow end-to-end training.

The approach from [46] is related to *System III*, which we will present in Chapter 6. Both systems use RL to learn a pose estimation pipeline including discrete choices. However, Brachmann *et al.* learn only a single discrete choice at the end of their pipeline, namely which of the hypotheses is to be selected as final estimate. This enables them to learn the prediction of object coordinates at an earlier stage of the pipeline in an end-to-end fashion. In contrast, we will describe how to train an iterative system to dynamically make repeated discrete choices with the goal of making best use of a restricted budget.

1.3.5 Direct Regression Methods

Finally, we want to mention methods that attempt to directly regress the pose from an observed image. This is a difficult task and the approach has until recently not been very influential with respect to object pose estimation. It remains to be seen whether this will change in the future.

Direct regression has been applied with limited success to the task of camera pose estimation [43, 76].

In [77], Mahendran *et al.* apply the method to extract the pose of an object from an RGB image. They assume that the bounding box of the object is known and use a CNN to process this patch and to do a regression of the 3D object orientation. Doumanoglou *et al.* [78] follow a similar approach, using the network to regress only the orientation of the object.

Very recently however, Xiang *et al.* [79] have proposed a state-of-the-art system that uses a CNN to directly predict a semantic segmentation of an image as well as the full 6D pose of objects. We believe that this line of research could bear a significant potential for the future.

1.4 Thesis Overview

In this section we want to provide a brief overview over the most important elements of the thesis. At the center of this thesis we will present three systems. All of them use probabilistic models and sampling techniques in different ways to address different aspects of the pose estimation problem. All three systems are based on the common framework presented by Brachmann *et al.* in [31]. Figure 1.4 provides an illustration and introduces the framework's main components. A more detailed description can be found in the following Chapter 2.

The thesis will address the online tracking task from Section 1.1.2 (*System I*) and the one-shot pose estimation task defined in Section 1.1.1 (*System II* and *System III*).

In *System I*, we will present a real-time PF-based system that uses a set of samples in pose space to represent and propagate uncertainty over time. In *System II* we use a CNN to assess the quality of pose hypotheses, using a probabilistic sampling-based approach during training. Finally, in *System III*, we attempt to directly learn an optimal search procedure that makes best use of a restricted computational budget. We will now give a brief summary of the these three systems.

1.4.1 System I: Pose Tracking

System I is a PF-based [32] adaptation of the one-shot pose estimation framework (see Figure 1.4) for 6D pose tracking.

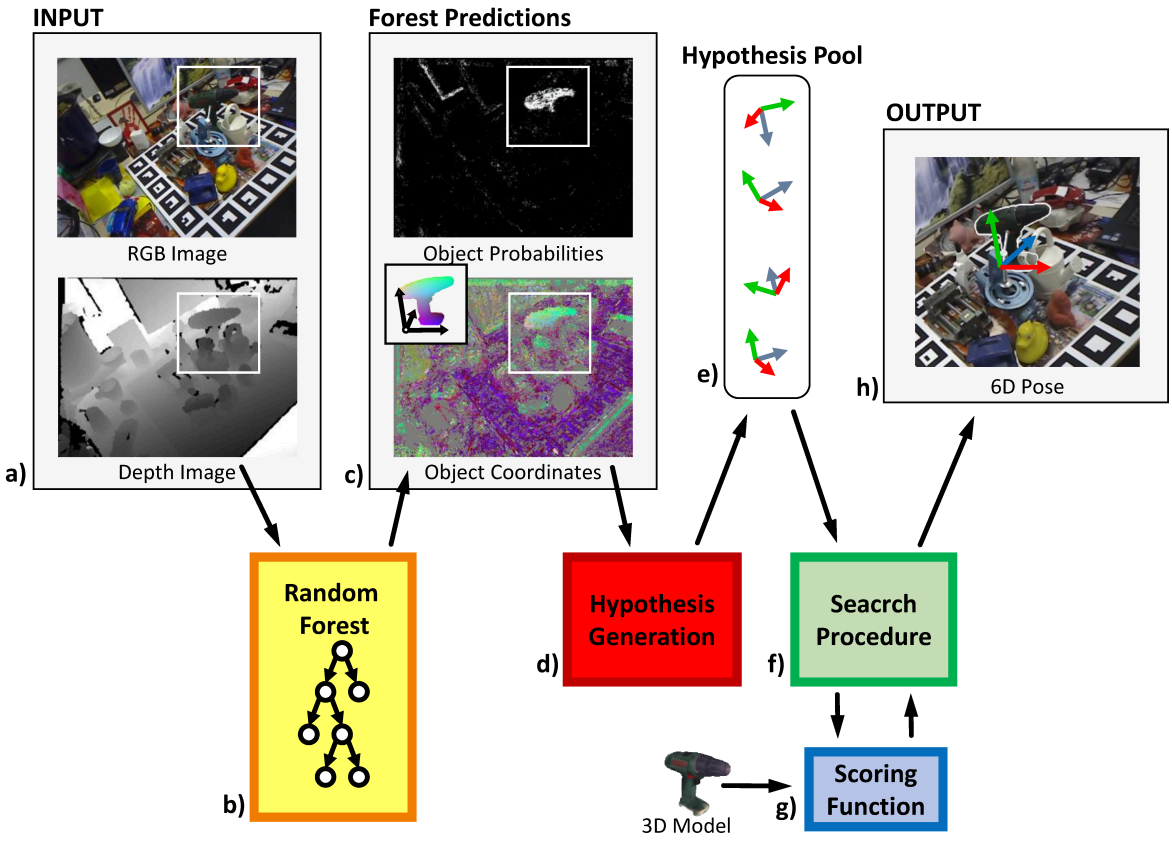


FIGURE 1.4: The pose estimation framework from [31]. The important components are shown as colored boxes. **a):** The system takes an RGB-D image as input. **b):** A random forest processes each pixel in the image by looking at the surrounding patch. **c):** It predicts two labels for every pixel. Top: Object probabilities, indicating whether the pixel is part of the object. Bottom: The position of the pixel in the local coordinate system of the object, termed object coordinates. Here, the predicted coordinates are mapped to the RGB cube for visualization. **d):** A sampling procedure uses the predictions to generate pose hypotheses. **e):** These hypotheses are summarized in a hypothesis pool. **f):** A search procedure takes the hypothesis pool as starting point and attempts to optimize a scoring function to find a good pose estimate. **g):** The scoring function judges the quality of a pose by comparing rendered and observed images. **h):** The highest scoring pose that was found is chosen as final estimate.

The PF is an established tracking approach. It models the knowledge about the pose at any point in time as probability distribution, represented by a set of samples in its *state space*. These samples are termed *particles*. In our case the state space corresponds to the 6D pose space. Whenever a new frame is processed the samples are updated to account for the newly made observation and a possible motion of the object that might have occurred in the mean time.

The PF is based on a probabilistic model consisting of two components: An *observation model* that describes the probability of observing a particular image, given a

particular state and a *motion model* that probabilistically describes the expected motion of the object. We use a variant of the scoring function from [31] (Figure 1.4 g) as basis of our observation model and combine it with a simple motion model, based on the assumption of smooth movements.

The fact that the pose space is 6D is challenging because high dimensional domains require a large number of samples to be covered adequately. It is common practice in PFs to reduce the number of required particles through an adequate *proposal distribution*. Proposal distributions are a part of the sampling scheme that can help to concentrate particles in the relevant areas. We follow this approach and use a specialized proposal distribution that is based on the hypothesis generation scheme (Figure 1.4 d) from [31].

1.4.2 System II: Learning Analysis-By-Synthesis

In *System II* we address the one-shot pose estimation problem. Again we use the framework from [31] as a starting point. We focus on improving the scoring function (Figure 1.4 g) of [31] to gain additional robustness with respect to occlusion.

The framework already provides some inherent stability in this respect because it is based on dense pixel-wise correspondences. Even when a part of the object is occluded the correspondences of the remaining visible part can still be used to create valid pose hypotheses.

The analysis-by-synthesis scoring function (Figure 1.4 g) used to assess the quality of the pose hypotheses is however vulnerable to occlusion. It performs a simple pixel-wise comparison of the observed images and images created by rendering a 3D model under a particular pose hypothesis.

We propose to replace this pixel-wise comparison with a CNN that takes rendered and observed images as input and outputs a score. Training such a system in a supervised fashion is not straightforward, since we have no ground truth scores available. Instead, we take a probabilistic approach and interpret the CNN as the energy function in an energy-based probabilistic model. Training our CNN as part of the energy-based model requires integration over pose space. We approximate the integral via a Monte Carlo sampling technique.

The final system is able to outperform the original approach from [31], in particular in images with high levels of occlusion.

1.4.3 System III: Pose Estimation on a Budget

In *System III* we go a step further. Instead of learning only a function to score hypotheses we propose to directly learn the *search process* (Figure 1.4 f) to find the correct pose while making optimal use of a computational budget.

The method from [31] uses a fixed algorithm to search for the correct pose. It refines the highest ranking hypotheses and then outputs the single highest ranking among the refined ones. Instead, we propose to train a system to dynamically distribute a predefined budget of refinement steps among hypotheses. The system works iteratively, choosing a new hypothesis for refinement in every step until its budget is exhausted. Training such a system in a principled end-to-end fashion is not trivial because choosing the next hypothesis from the pool for refinement is a discrete non-differentiable operation.

We find that a stochastic *policy gradient* method [80] from the field of RL can provide an adequate answer to this problem. The core idea is to think of discrete choices as non-deterministic and governed by probability distributions. During training, we can optimize the expected number of correctly classified poses using gradient descent over the parameters of said probability distributions.

1.5 Contributions

Here, we want to give a short summary of the main contributions presented in this thesis. In this work, we present three major contributions:

- We present a PF method for real-time 6D pose tracking-based on the framework of [31].
- We present a probabilistically motivated system that learns to compare rendered and observed images to assess the quality of a pose hypothesis. When we use it in the framework of [31], we are able to significantly improve accuracy, in particular for highly occluded objects.
- We present an RL-based procedure that learns to dynamically distribute a computational budget in a search for the correct pose. The procedure includes non-differentiable discrete choices that can be trained via stochastic policy gradient.

1.6 Structure of the Thesis

This work is structured into seven chapters. The following two chapters contain different preliminary considerations. In Chapter 2 we will give a detailed description of the framework from [31], which is the foundation of the systems we will present afterwards. In Chapter 3 we will take a closer look at the 6D pose space. We will in particular discuss the mathematical peculiarities which are to be considered when working with probability distributions in pose space.

The Chapters 4 to 6 will present our main contributions: In Chapter 4 we will describe *System I* (published in [48]), the PF-based real-time pose tracking system. In

Chapter 5 (published in [81]), we will discuss *System II*, the replacement of the simple scoring function (Figure 1.4 g) from [31] with a learned function, implemented by a CNN and trained using a probabilistic perspective. In Chapter 6 (published in [82]) we will go a step further and describe *System III*, an RL-based method replacing not only the scoring function, but the entire search procedure (Figure 1.4 f).

Finally, in Chapter 7 will conclude with a discussion and an outlook into possible directions of future research.

Chapter 2

Pose Estimation Framework

Contents

In this chapter, we will give a description of the pose estimation framework from [31], which provides the background for the systems we will present in Chapters 4, 5, and 6. The framework addresses the one-shot pose estimation problem as described on Section 1.1.1. It takes an RGB-D image $\boldsymbol{x}$ as input and outputs an estimated pose $\hat{H}$ for a known object $c \in C$, which is part of a set of known objects C. Let us remember the general scheme, depicted in Figure 1.4, with its four major components:

- In a first step, the image is processed by a *random forest*, which provides pixel-wise predictions for two things: (i) The object probabilities, *i.e.* an estimated probability that the pixel belongs to the particular object c. (ii) The object coordinates, *i.e.* the position of the pixel in the local coordinate system of the object.
- In a second step, a *sampling procedure* uses the forest prediction to generate a pool of pose hypotheses $\boldsymbol{H} = (H_1, \ldots, H_N)$.
- Then, a *search procedure* takes the hypothesis pool as starting point.
- The goal of the search is to optimize a *scoring function*, assessing the quality of pose hypotheses. The best scoring pose is output as the final estimate $\hat{H}$.

In the following sections we will take a closer look at each of these components. We will conclude the chapter by introducing a simplified notation in the context of this framework that will be used in the remaining part of the thesis.

2.1 Random Forest

The authors use a random forest [68], denoted by $\mathcal{T}$. It consists of $|\mathcal{T}| = 3$ trees. Each tree j independently processes each pixel i. At every split node, the tree performs a feature test on a patch surrounding the pixel. Depending on the outcome, the pixel is routed to the left or right child node. The authors apply the basic feature tests from [83]. These tests use two pixel offsets relative the central pixel of the patch and calculate the difference in the depth and color intensities at the two corresponding locations. If the difference lies below or above a threshold, the test result is positive or negative, respectively. As in [83], the authors use the depth value at the central pixel to adjust the offsets accordingly, using larger offsets for close pixels and smaller offsets for pixels that are further away.

The trees can be trained using recorded images or rendered images of the object. Training images are annotated with the true segmentation of the object and the ground truth object coordinates for every pixel. Additionally generic training images containing no object from C are used to represent the appearance of the background.

Training is done in a two step procedure. First, the authors train the structure and feature tests at each node via a proxy classification problem. They discretize the space of object coordinates for each object $c \in C$ into $5 \times 5 \times 5 = 125$ regular bins and add an additional bin for background pixels. This allows for the use of the standard information gain target over the resulting $125 \times |C| + 1$ classes. The authors combine it with the randomized training procedure from [84].

In a second step, after the training of the tree structure is completed, the authors again feed training pixels from all objects and from the background images through the trees and store the ground truth object coordinates at every leaf. For every object c and at every leaf, they then perform *mean-shift* clustering [85] on the ground truth object coordinates, using an isotropic Gaussian kernel with standard deviation of 25mm. They store the positions of the largest mode of every object at each leaf together with the empirical distribution over the objects, *i.e.* the percentage of training pixels at that leaf which belong to the object c.

During testing, each pixel i is processed by each tree j. The pixel is routed through the tree's nodes and ultimately ends up in one of the tree's leaf nodes. The tree then outputs the information about the object stored at the leaf: The position $\boldsymbol{y}^j_{i,c}$ of the largest mode of object coordinates and the empirical probability $p^j_{i,c}$ that the pixel belongs to the object. The predicted probabilities from all trees are combined to a single

object probability $p_{c,i}$ using

$$p_{c,i} = \frac{\prod_{j=1}^{|\mathcal{T}|} p_{i,c}^{j}}{\left(\sum_{c' \in C} \prod_{j=1}^{|\mathcal{T}|} p_{i,c'}^{j}\right) + \prod_{j=1}^{|\mathcal{T}|} p_{i,\text{bg}}^{j}}, \tag{2.1}$$

were $p_{i,\text{bg}}^{j} = 1 - \sum_{c \in C} p_{i,c}^{j}$ is the empirical probability predicted by tree j that pixel i belongs to the background.

2.2 Hypothesis Generation

The authors use a RANSAC-like system to generate a hypothesis pool $\boldsymbol{H}$ from sampled pixels. To find a pose hypothesis they sample three pixels. Each pixel provides a 3D-3D correspondence between the camera coordinates $\boldsymbol{d}_i$ and the predicted coordinates $\boldsymbol{y}_{i,c}^{j}$ in the local coordinate system of the object. The 3D-3D correspondences from all three pixels are then combined to generate a pose hypothesis.

Let us now take a look at this procedure in detail. To generate a pose hypothesis, the authors start by sampling a pixel i_1. The pixel is drawn from a probability distribution with probability mass function proportional to $p_{c,i}$. Thus, pixels with higher probability of belonging to the object have a higher probability of being selected. Then, they draw two additional pixels i_2 and i_3 again using a probability mass function proportional to $p_{c,i}$. This time however, they consider only pixels in a square window around the original pixel i_1. The rationale being that pixels, which are too far apart can impossibly belong to the same object. The window size is calculated as $f\delta_c/d_{i_1}$, where f is the focal length of the camera and δ_c is the diameter of the object.

After determining three pixels i_1,i_2,i_3, the authors randomly select tree indexes j_1,j_2,j_3 to obtain the correspondences $(\boldsymbol{d}_{i_1}, \boldsymbol{y}_{i_1,c}^{j_1})$, $(\boldsymbol{d}_{i_2}, \boldsymbol{y}_{i_2,c}^{j_2})$, and $(\boldsymbol{d}_{i_3}, \boldsymbol{y}_{i_3,c}^{j_3})$. Then, based on said correspondences, a pose hypothesis H is found via the *Kabsch algorithm* [86], a well established closed form technique to estimate a pose from a set of 3D-3D correspondences.

The hypothesis H is accepted and included in the hypothesis pool $\boldsymbol{H}$ if the Euclidean distances between the observed camera coordinates $\boldsymbol{d}_{i_1}$, $\boldsymbol{d}_{i_2}$, $\boldsymbol{d}_{i_3}$ and the transformed predicted object coordinates $H(\boldsymbol{y}_{i_1,c}^{j_1})$, $H(\boldsymbol{y}_{i_2,c}^{j_2})$, $H(\boldsymbol{y}_{i_3,c}^{j_3})$ are below 5% of the object diameter. The process is repeated until a pool size of $N = 210$ is reached.

2.3 Search Procedure

The framework uses a simple search procedure starting from the generated hypothesis pool $\boldsymbol{H}$ and trying to optimize the scoring function (see next Section 2.4). The procedure begins by calculating the score for each of the hypotheses in the pool. The

25 best hypotheses are then subject to the refinement procedure, described in Section 2.3.1. After the refinement is complete, each hypothesis is again processed by the scoring function. The hypothesis with the best score is returned as final estimate $\hat{H}$. The entire procedure is depicted in Figure 2.1.

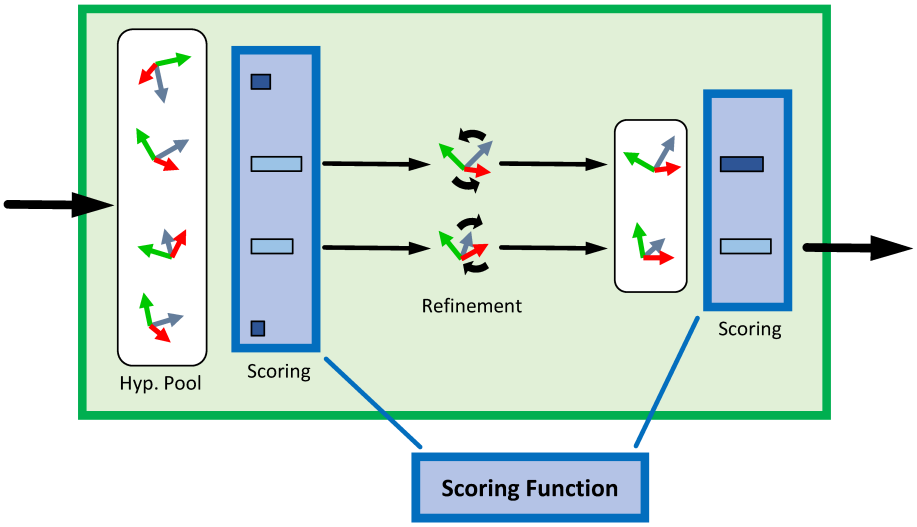


FIGURE 2.1: The search procedure used in [31]: The procedure takes the hypothesis pool as starting point and finds a pose estimate $\hat{H}$ through a process of scoring and refinement. See Section 2.3.

2.3.1 Refinement

The refinement procedure is an iterative process similar to the hypothesis generation described in the previous Section 2.2. Given a particular pose H that is to be refined, the object is rendered under the pose to obtain a mask $M_c(H)$, which is the set of pixels belonging to the rendered object, excluding pixels with no valid recorded depth information d. The procedure now evaluates each pixel $i \in M_c(H)$ and calculates the minimum of Euclidean errors

$$e_i = \min_{j \in \{1 \dots |\mathcal{T}|\}} \left(|H(\boldsymbol{y}^j_{i,c}) - \boldsymbol{d}_i| \right) \tag{2.2}$$

between the transformed object coordinate and the recorded depth information. If e_i is below a threshold of 20mm, the pixel i is counted as inlier pixel and the correspondence $(\boldsymbol{d}_i, \boldsymbol{y}^{\tilde{j}}_{i,c})$ is added to a set of correspondences. Here $\tilde{j}$ is the tree that led to the lowest Euclidean error $|H(\boldsymbol{y}^{\tilde{j}}_{i,c}) - \boldsymbol{d}_i|$. The set of correspondences is then used to derive an updated hypothesis via the Kabsch algorithm. After every iteration the scoring function is applied again. The entire process is repeated iteratively until the scoring function of the updated pose is no longer improving.

2.4 Scoring Function

The scoring function plays an important role, as it is the objective function the system attempts to optimize. To assess the quality of a pose hypotheses H the function renders the object under the pose to compare the results with the observed depth values, as well as with the predicted object coordinates and object probabilities. The function is a weighted sum of three components:

$$E(H) = \lambda_{\text{depth}} E_{\text{depth}}(H) + \lambda_{\text{obj}} E_{\text{obj}}(H) + \lambda_{\text{coord}} E_{\text{coord}}(H) \tag{2.3}$$

The function $E_{\text{depth}}(H)$ compares a rendered depth image with the observed depth image $\boldsymbol{d}$. The function $E_{\text{obj}}(H)$ compares a rendered mask with the predicted object probabilities $p^j_{i,c}$. Finally, the function $E_{\text{coord}}(H)$ compares a rendered image of object coordinates with the predicted object coordinates $\boldsymbol{y}^j_{i,c}$. The parameters λ_{depth}, λ_{obj}, and λ_{coord} control the importance of the different components. Let us now examine the components in detail.

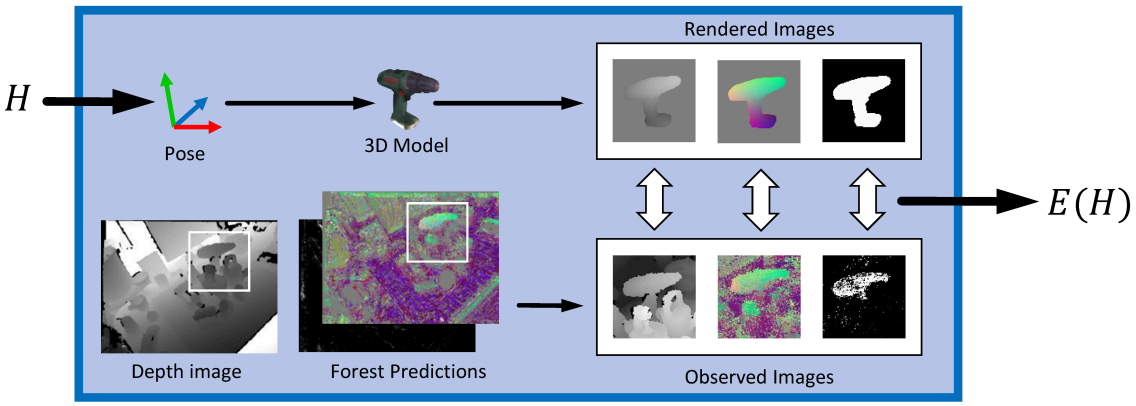


Figure includes images and models from the dataset by Hinterstoisser *et al.* [14], published in 2012 under a CC BY 4.0 Licence.

FIGURE 2.2: The scoring function as it is used in [31]: To assess the quality of a pose hypothesis H the function renders the 3D model of the object under H and then compares rendered and observed images. The final value is a weighted sum of three pixel-wise distance functions.

2.4.1 Depth Component

This component compares the observed depth image $\boldsymbol{d}$ and the rendered depth image $\hat{\boldsymbol{d}}(H)$. It is defined as the sum

$$E_{\text{depth}}(H) = \frac{\sum_{i \in M_c(H)} f\left(\boldsymbol{d}_i, \hat{\boldsymbol{d}}_i(H)\right)}{|M_c(H)|} \tag{2.4}$$

over the set of pixels $M_c(H)$ that are part of the object in the rendered images. The pixels for which no depth information is provided by the sensor are additionally excluded. The function

$$f\left(\boldsymbol{d}_i, \acute{\boldsymbol{d}}_i(H)\right) = \min\left(|\boldsymbol{d}_i - \acute{\boldsymbol{d}}_i(H)|, \tau_d\right)/\tau_d \tag{2.5}$$

is the normalized and truncated Euclidean distance between the rendered and observed camera coordinates at the pixel.

2.4.2 Object Component

The object component is again a sum over the pixels in $M_c(H)$. It is defined as

$$E_{\text{obj}}(H) = \frac{\sum_{i \in M_c(H)} \sum_{j=1}^{|\mathcal{T}|} -\log(p_{i,c}^j)}{|M_c(H)|}. \tag{2.6}$$

2.4.3 Coordinate Component

The component compares the predicted object coordinates $\boldsymbol{y}_{i,c}^j$ and rendered object coordinates $\acute{\boldsymbol{y}}_{i,c}(H)$. It is defined as

$$E_{\text{coord}}(H) = \frac{\sum_{i \in M_c^L(H)} \sum_{j=1}^{|\mathcal{T}|} g\left(\boldsymbol{y}_{i,c}^j, \acute{\boldsymbol{y}}_i(H)\right)}{|M_c^L(H)|}. \tag{2.7}$$

Note that a different mask $M_c^L(H)$ is used here. It is identical to $M_c(H)$, except for additionally excluding pixels where $p_{c,i} < \tau_{\text{pc}}$. The reason is that pixels with predicted $p_{c,i}$ below the threshold τ_{pc} are found to have unreliable object coordinate predictions. The function

$$g\left(\boldsymbol{y}_{i,j}, \acute{\boldsymbol{y}}_i(H)\right) = \min\left(|\boldsymbol{y}_{i,j} - \acute{\boldsymbol{y}}_i(H)|, \tau_y\right)/\tau_y \tag{2.8}$$

performs a robust comparison of the rendered and predicted object coordinates at a pixel i.

2.5 Simplified Notation

In the remaining part of the thesis we will use a notation that is simplified in two respects:

(i) The three systems we will present are only concerned with one object c at a time. We will omit the reference to the object c to improve readability.

(ii) The focus of the three systems lies not on the prediction of object coordinates and object probabilities, but on later parts of the framework. We will thus see the application of the forest as a kind of preprocessing and summarize all forest predictions together with the original image $\boldsymbol{x}$ as *observation* and denote it as $\boldsymbol{z}$.

Chapter 3

Probability Distributions in Pose Space

Contents

In this chapter we will take a closer look at the mathematical peculiarities of the pose space. Since we follow a probabilistic sampling-based approach we will focus in particular on the problem of defining probability distributions over poses.

Let us remember the definition of a pose we gave in Section 1.1.1: A pose is a pair

$$H = (R, \boldsymbol{v}), \tag{3.1}$$

of a rotation R and a translation $\boldsymbol{v}$. We consider R to be a 3×3 rotation matrix [1] and $\boldsymbol{v}$ as to be a simple 3D vector. Together, R and $\boldsymbol{v}$ describe a transformation

$$H(\boldsymbol{y}_i) = \boldsymbol{v} + R\boldsymbol{y}_i, \tag{3.2}$$

mapping a point $\boldsymbol{y}_i$ in the local coordinate system of the object to its representation in the coordinate system of the camera. This kind of transformation is generally referred to as *proper rigid transformation*, as it preserves the distance between any two transformed points. The pose space is also referred to as the space of proper rigid transformations and is often denoted by $SE(3)$. This space has a particular mathematical structure known as *Lie group* [87].

[1] We will give a discussion of rotation matrices in Section 3.1.1

In the course of this chapter we will not consider the structure of $SE(3)$ directly. Instead, we will address the rotation and translation components separately, as done e.g. in [88]. The parametric distributions we will use in this thesis, will all model rotation and translation as independent variables. This allows us to describe a joint probability density as the product of two distributions over rotation and translation.

The space of translations is a vector space and identical to the 3D Euclidean space $\mathbb{R}^3$. Here, we can define probability distributions without any additional considerations. The space of rotations on the other hand is not handled as easily. Defining a probability distribution over rotation matrices is not a trivial task. Similarly to $SE(3)$ itself, the space of rotations has the internal structure of a Lie group. It is denoted by $SO(3)$.

In the following Section 3.1 we will discuss the space of rotations and how to construct probability distributions in it. Finally, in Section 3.2 we will address the pose space itself and describe a class of joint probability distributions over rotations and translations, modeled as a product.

3.1 Rotations

In this section, we will take a look at rotations in 3D space. We will start by exploring the different ways to represent rotations and then discuss the internal structure of $SO(3)$ as a Lie group. We will briefly consider the problem of averaging in the Lie group $SO(3)$. and finally, discuss probability distributions over $SO(3)$.

3.1.1 Representing Rotations

Rotations in 3D Euclidean space can be represented in different ways, each of them having a number of advantages and disadvantages. Here, we will discuss only the most important representations with respect to this thesis. A survey on this topic can be found in [89]. In [90] Shuster gives a more in depth discussion.

Matrix Representation

It is possible to describe any 3D rotation as a 3×3 orthogonal matrix R with determinant $\det(R) = 1$. Such a matrix will be referred to as *rotation matrix*. We can construct matrices for rotations of arbitrary angles around the three standard axes in the following way:

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}, R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix}, R_z(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.3}$$

Rotation matrices have a number of advantages:

- The representation is unique in the sense that every rotation is represented by exactly one rotation matrix.
- A rotation, represented as matrix, can be applied to a vector by simple vector-matrix multiplication.
- Two rotations can be combined, as if being executed subsequently, by simple matrix multiplication.
- The direction of a rotation can be inverted by inverting the matrix. Since rotation matrices are orthogonal, this is equivalent to transposing the matrix $R^{-1} = R^\intercal$.
- We can calculate the angle of a rotation as R as

$$\psi = \arccos\left(\frac{1}{2}(\text{tr}(R) - 1)\right), \tag{3.4}$$

  where $\text{tr}(R)$ is the trace of the matrix R
- We can calculate the *difference rotation* between two rotation matrices R_0 and R_1 as $R_0 R_1^{-1}$.
- We can define a meaningful distance metric between rotations R_0 and R_1 as

$$\psi_{R_0,R_1} = \psi_{R_1,R_0} = \arccos\left(\frac{1}{2}\left(\text{tr}(R_0^{-1}R_1) - 1\right)\right). \tag{3.5}$$

  The resulting ψ_{R_0,R_1} is the angle of the difference rotation between R_0 and R_1.

On the other hand rotation matrices also come with some disadvantages: They require nine parameters even though rotations have only three degrees of freedom. Additionally, it is not straightforward to define probability distributions over rotation matrices.

Nevertheless, due to their numerous advantages, we will use rotation matrices as our standard representation during this thesis.

Euler Angles

Euler angles describe any rotation in 3D as a sequence of three rotations around different axes. Any 3D rotation can be represented by a 3-tuple of such angles. Different conventions about the particular order of axes and rotations exist [89]. One approach is to use angles γ, β, α to parametrize subsequent rotations around the three axes z, y, x in this order.

We can convert the Euler angles, described above, into matrix representation by multiplying the matrices from Eq. 3.3,

$$R = R_x(\alpha)R_y(\beta)R_z(\gamma), \tag{3.6}$$

thus sequentially executing rotations around the three axes.

A big advantages of the Euler angle representation is that it requires only three parameters and that all possible configurations of these parameters describe valid rotations.

The downside of Euler angles lies in the interdependence of the parameters. In an extreme case, when two of the three rotation axes align the so-called *gimbal lock* occurs. In such a case changing any of the two corresponding angles has an identical effect. Euler angles are not unique, as multiple sets of angles can represent the same 3D rotation.

In this thesis, we will not use Euler angles directly to represent rotations. We will however make use of the idea and Eq. 3.6.

Axis-Angle and Euler Vector

It is possible to describe any element of $SO(3)$ as a rotation around a single axis. The *axis angle* representation consists of a unit vector defining a rotation axis and a scalar value describing the angle of the rotation.

The *Euler vector*, also known as *rotation vector* is a condensed form of this representation. It describes a rotation using a single 3D vector $\boldsymbol{\omega}$. The orientation of this vector corresponds to the axis of rotation. The Euclidean length of the vector $|\boldsymbol{\omega}|$ corresponds to the angle of the rotation.

The Euler vector combines two major advantages: First, it consists of only three parameters and as with the Euler angles each possible combination of the values corresponds to a valid rotation. It is thus possible to define probability distributions over the parameter space. Second, even though the Euler vector is not unique, we can achieve a unique one to one representation by restricting the range of allowed parameters. Since the length of the vector $|\boldsymbol{\omega}|$ corresponds to the angle of rotation, the same rotation can be represented by multiple vectors. We can add or subtract any integer multiple of 2π to the length a vector, and it will still represent the same rotation. Indeed we require only the set of vectors with $|\boldsymbol{\omega}| < \pi$ [2] to represent all possible rotations [91]. Any rotation by a larger angle can equally be represented, by a rotation in the opposite direction.

[2] Strictly speaking the rotations with an angle of exactly π or 180 degrees are not covered in the volume $|\boldsymbol{\omega}| < \pi$. We will not address this issue, because the probability of encountering a rotation of this exact angle is zero for any probabilistic model we will consider in this thesis.

Euler vectors are deeply connected with the matrix representation, we will address this connection explicitly in Section 3.1.2. In this thesis, Euler vectors will play an important role for the definition of probability distributions over rotations.

3.1.2 Lie Group of Rotations

The set of 3D rotations has a mathematical structure referred to as Lie group. The particular Lie group of 3D rotations is called the *special orthogonal group*, denoted as $SO(3)$. Gallier and Quaintance [87] give thorough introduction and discussion on the topic of Lie groups including $SO(3)$. A very concise summary on the use of $SO(3)$ for computer vision can be found in [92].

Lie groups are groups that are at the same time *differentiable manifolds* [87]. We will first discuss $SO(3)$ as a group and then look at its property as a manifold.

SO(3) as Group

Let us first discuss the property of being a group with respect to $SO(3)$. A group consists of a set of elements G and a group operation $a \cdot b$ that can be applied to any two of the elements $a, b \in G$. In the case of $SO(3)$ the elements correspond to 3D rotations. The group operation corresponds to the successive application of the rotations. A group has to fulfill the following requirements [93]:

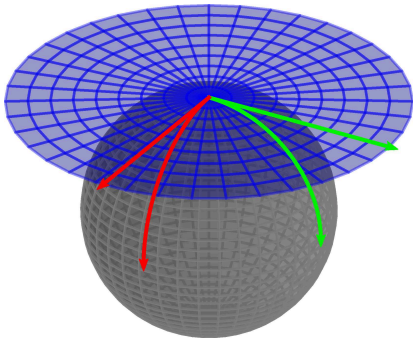
- **Closure:** A group has to be closed with respect to its group operation. This means that every group element a combined with any other group element b via the group operation $a \cdot b$, will result in another member of the group. Considering $SO(3)$, this requirement is fulfilled because any successive application of two rotations will result in another rotation.
- **Associativity:** For any group elements a, b, c it has to hold that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. This is true for the successive application of rotations.
- **Identity element:** There has to be an identity element e, such that $e \cdot a = a = a \cdot e$ for any element a. There is such an element in the $SO(3)$. Namely, a rotation of angle zero represented as identity matrix.
- **Inverse elements:** For every element a, there has to exist an inverse element b, such that $a \cdot b = e = b \cdot a$. In the group of 3D rotations such an element can be found for each rotation by inverting the direction of rotation.

SO(3) as Manifold

Let us now look at the properties of $SO(3)$ as a smooth manifold. Smooth manifolds are spaces that are locally similar to the Euclidean space $\mathbb{R}^n$ and allow for the use of

calculus [94]. Intuitive examples of smooth manifolds would be the curved surfaces in 3D Euclidean space, or curved lines in the plane. We will use the sphere in $\mathbb{R}^3$ as an example illustrate the points we will make about the manifold $SO(3)$. As the 2D surface of the sphere is embedded in the 3D Euclidean space, the 3D manifold of rotation matrices $SO(3)$ is embedded in the 9D space of 3×3 real valued matrices [95]. In both cases the surface of the manifold is curved. This makes it difficult to perform operations like interpolation and averaging. It also makes it hard to define probability distributions directly on the manifold.

There is however a tool that can help with these issues. In the case of the sphere as well as for $SO(3)$ it is possible to define *tangent spaces*. A tangent space is a real valued vector space of the same dimension as the manifold. It will allow us to locally perform vector arithmetic, like the addition and subtraction of vectors. A tangent space can be attached at any point on the manifold. Using the example of the sphere in $\mathbb{R}^3$, a tangent space coincides with the 2D Euclidean space $\mathbb{R}^2$. Figure 3.1 provides an illustration. Each point in the tangent space can be mapped to a point on the manifold. In the context of this thesis we will consider tangent spaces that use a particular kind of map, called the *exponential map* for this connection.

a)  b) 

Flag of the United Nations taken from Wikipedia, public domain.

FIGURE 3.1: Manifolds and tangent spaces: **a):** A tangent space of the sphere in $\mathbb{R}^3$ at its north pole. The sphere is a manifold. Its tangent space is a vector space corresponding to $\mathbb{R}^2$. Each point in the tangent space can be mapped to a point on the manifold. The map from this particular tangent space to the manifold is called exponential map. The origin of the tangent space is mapped to the north pole. If one were to move away from the origin in a straight line this would coincide with moving south on the sphere. Note that the south pole of the sphere corresponds to a circle in the tangent space. **b):** The flag of the united nations is based on an *azimuthal equidistant projection* of the earth, which corresponds to the exponential map [96].

Tangent Space at the Identity Element

We will now describe how to construct a tangent space for the Lie group $SO(3)$. First, we will consider a tangent space at the identity rotation. The tangent space at the

identity element of a Lie group is called *Lie algebra* [87]. The Lie algebra of $SO(3)$ is denoted by $\mathfrak{so}(3)$.

We can represent the identity rotation as the 3×3 identity matrix. The basis vectors of the tangent space at this location are given by the derivatives of the matrix with respect to rotations around the three standard axis of the coordinates system [87, 92]. This corresponds to the derivatives of the Euler Angle parametrization given in Equation 3.6 with respect to the three angles at the position $\alpha = 0, \beta = 0, \gamma = 0$. The resulting basis of our tangent space consists of three matrices:

$$G_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, G_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, G_3 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{3.7}$$

Any point $\boldsymbol{\omega}_\times$ in the tangent space can now be expressed as a linear combination resulting in a skew symmetric matrix

$$\boldsymbol{\omega}_\times = \omega_1 G_1 + \omega_2 G_2 + \omega_3 G_3 = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}. \tag{3.8}$$

We can now represent any rotation as a 3D vector $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^\intercal$ of coefficients. Interestingly, this vector coincides exactly with the Euler vector discussed in Section 3.1.1 [92, 97]. To simplify notation, we will refer to both, the skew symmetric matrix $\boldsymbol{\omega}_\times$ and the corresponding Euler vector $\boldsymbol{\omega}$, as elements of the tangent space.

Exponential Map and Logarithmic Map

Let us now consider the mappings between the tangent space and the manifold, *i.e.* between the Lie algebra $\mathfrak{so}(3)$ and the Lie group $SO(3)$. Since the vector $\boldsymbol{\omega}$ of coefficients is equivalent to the Euler vector, these maps will also describe the conversion between Euler vector representation and matrix representation.

To map an element $\boldsymbol{\omega}_\times$ of the Lie algebra to the Lie group, we can use the following equation [87, 92]. It is called the exponential map of $SO(3)$:

$$R = \exp(\boldsymbol{\omega}_\times) = \mathbf{I} + \frac{\sin(\psi)}{\psi}\boldsymbol{\omega}_\times + \frac{1 - \cos(\psi)}{\psi^2}\boldsymbol{\omega}_\times^2, \tag{3.9}$$

where $\psi = |\boldsymbol{\omega}|$ is the angle of rotation, equal to the vector norm of the Euler vector.

The inverted mapping of an element R from the Lie group to the Lie algebra is called the *logarithmic map*. It can be performed [87, 92] by first finding the angle ψ of rotation via Eq. 3.4. If $\psi \neq 0$ we can proceed to calculate the skew symmetric matrix

as

$$\boldsymbol{\omega}_{\times} = \log(R) = \frac{\psi}{2\sin(\psi)}\left(R - R^{\intercal}\right). \tag{3.10}$$

We will use the following simplified notation with respect to the exponential and logarithmic map: Since the Euler vector $\boldsymbol{\omega}$ can be trivially derived from the corresponding skew symmetric matrix $\boldsymbol{\omega}_{\times}$ and vice versa through Eq. 3.8, we will use the same expressions $\log(\cdot)$ and $\exp(\cdot)$ to refer to the maps with respect to skew symmetric matrices and with respect to the Euler vector representation.

Tangent Spaces at Other Locations

Let us now consider how to use a tangent space at a general location R_0 other than the identity element. We will follow [98] [3] in their definition of functions for the exponential and logarithmic map corresponding to the tangent space at particular elements.

We will use $\boldsymbol{\omega}_{R_0}$ to denote an element of the tangent space at a particular location R_0 on $SO(3)$. To map $\boldsymbol{\omega}_{R_0}$ to the corresponding element R on the manifold $SO(3)$ we can use the function

$$R = \exp_{R_0}(\boldsymbol{\omega}_{\boldsymbol{R_0}}) = \exp(\boldsymbol{\omega}_{\boldsymbol{R_0}})R_0. \tag{3.11}$$

To map an element R of $SO(3)$ to the corresponding element $\boldsymbol{\omega}_{R_0}$ of the tangent space at location R_0, we can use

$$\boldsymbol{\omega}_{R_0} = \log_{R_0}(R) = \log(RR_0^{-1}). \tag{3.12}$$

By combining Eq. 3.11 and Eq. 3.12 we can define a function $f_{R_0 \to R_1}(\boldsymbol{\omega}_{R_0})$ that maps an element from one tangent space at R_0 to the corresponding element in another tangent space at a different location R_1:

$$\boldsymbol{\omega}_{R_1} = f_{R_0 \to R_1}(\boldsymbol{\omega}_{R_0}) = \log\left(\exp(\boldsymbol{\omega}_{R_0})R_0{R_1}^{-1}\right). \tag{3.13}$$

Note that as long as we consider only elements $|\boldsymbol{\omega}_{R_0}| < \pi$, the map $f_{R_0 t R_1}$ is a one-to-one mapping and invertible as

$$\boldsymbol{\omega}_{R_0} = f^{-1}_{R_0 \to R_1}(\boldsymbol{\omega}_{R_1}) = f_{R_1 \to R_0}(\boldsymbol{\omega}_{R_1}) = \log\left(\exp(\boldsymbol{\omega}_{R_1})R_1{R_0}^{-1}\right). \tag{3.14}$$

3.1.3 Averaging Rotations

Considering the manifold structure of $SO(3)$ averaging is not trivial. Rotations can be averaged in a variety of different ways.

[3] Note that the definition in [98] uses a multiplication on the left in the exponential map while we use a multiplication on the right. This difference leads to tangent spaces that are rotated around their origin, but otherwise identical [92].

An overview on the topic can be found e.g. in [95, 99, 100]. A rotation average can be found by calculating the matrix mean and projecting it back onto the manifold [99]. Other methods find an average by iteratively solving a minimization problem [95].

In this thesis we will always use an approximate solution to the following matrix-based mean as described in [100]. Given a set of rotation matrices $\boldsymbol{R} = \{R_1, \dots R_{|\boldsymbol{R}|}\}$, the mean rotation is defined as

$$R_{\text{mean}} = \underset{R}{\operatorname{argmin}} \sum_{i=1}^{|\boldsymbol{R}|} \psi_{R,R_i}^2, \tag{3.15}$$

with the angular distance ψ_{R,R_i} defined according to Eq. 3.5. The approximate solution to Eq. 3.16 can be found via *Singular Value Decomposition* (SVD), by calculating $UDV = \sum_{i=1}^{|\boldsymbol{R}|} R_i$ and

$$R_{\text{mean}} \approx USV, \text{ where } S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(U)\det(V) \end{pmatrix}. \tag{3.16}$$

3.1.4 Probability Distributions over Rotations

There exists a variety of different approaches of modeling probability distributions over Lie groups. A straightforward approach with respect to the generation of samples, is the definition of probability densities in the tangent spaces, e.g. [92, 101, 102]. Many techniques however, such as importance sampling, additionally require the evaluation and comparison of probability densities. It is not trivial to achieve this in a way, doing justice to the Lie group structure of $SO(3)$. Many authors take a measure theoretic approach [91, 103–105], and define probability distributions directly on the manifold. Densities can be defined with respect to the *Haar measure* [106], which acts as "uniform distribution" on $SO(3)$ [91].

In the subsequent sections, we will take the following approach: We will begin by describing distributions as density functions in the tangent spaces of $SO(3)$. Additionally, we will take special precautions when comparing two densities defined in different tangent spaces. Following this approach, we will introduce a useful class of probability distributions on $SO(3)$, called *uniform axis random spin* (UARS) distributions [91]. Finally, our approach will coincide with the measure theoretic perspective and will lead us to equivalent equations for the comparison of UARS densities, as the Haar measure does.

Probability Distributions in Tangent Space

One of the important applications of the Lie algebra $\mathfrak{so}(3)$ its ability to easily define probability distributions over rotations. Since $\mathfrak{so}(3)$ is a vector space, we can handle distributions just as we would normally do in $\mathbb{R}^3$. By using the exponential map from Eq. 3.9, the elements sampled from a distribution in a tangent space can be mapped to $SO(3)$, thus indirectly defining a distribution over rotations. It would be possible to define all probability distributions we require in the tangent space at the identity element. This was effectively done in [102].

A problem with this approach is that a distribution defined in the tangent space at the identity will be distorted as it is mapped to the manifold via the exponential map. The situation is comparable to the inevitable distortions in a 2D map of planet earth. Consider the map in Figure 3.1 b). Imagine drawing a circle of a specified radius on the map and consider then the corresponding shape on a 3D globe. The resulting shape on the globe depends on the position of the circle. A circle drawn around the center of the map corresponds to a circular shape on the globe. If we draw a circle of equal radius shifted towards the map's edge, e.g. in Antarctica, the corresponding shape on the globe will become smaller and distorted.

Similarly, if we define a distribution in $\mathfrak{so}(3)$, it will undergo a deformation when mapped to $SO(3)$. This distortion will be less severe for distributions centered close to the origin and more extreme for distributions that are centered further away from the origin. A visualization of the effect is shown in Figure 3.2. Note that the distorted distributions (in the top row) are more compact, as it would be the case for the circles drawn on the map.

A solution to this problem is to always define a distribution in the tangent space located at the distribution center. This is the approach of most literature, e.g. [92, 101, 104], and we will follow it in this thesis as well.

There are however some pitfalls when using probability distributions defined in different tangent spaces. While generally unproblematic with respect to sampling, special care has to be taken for the comparison of probability densities defined in different tangent spaces.

Consider the following problem: Two probability distributions, with density $p^a_{R_0}(\boldsymbol{\omega}_{R_0})$ and $p^b_{R_1}(\boldsymbol{\omega}_{R_1})$, are defined in tangent spaces at the positions R_0 and R_1 respectively. We assume that the support of the distributions is limited to the volume $|\boldsymbol{\omega}_{R_0}| < \pi$, $|\boldsymbol{\omega}_{R_1}| < \pi$, thus covering all possible rotations once. We are now interested in comparing the probability densities for a particular location R on the manifold, *i.e.* we want to find the ratio of the two densities at R. Such a computation is required e.g. for the importance sampling techniques, as the one we will use in Chapter 4.

Note that we cannot simply use Eq. 3.12 and compare $p^a_{R_0}(\log_{R_0}(R))$ with

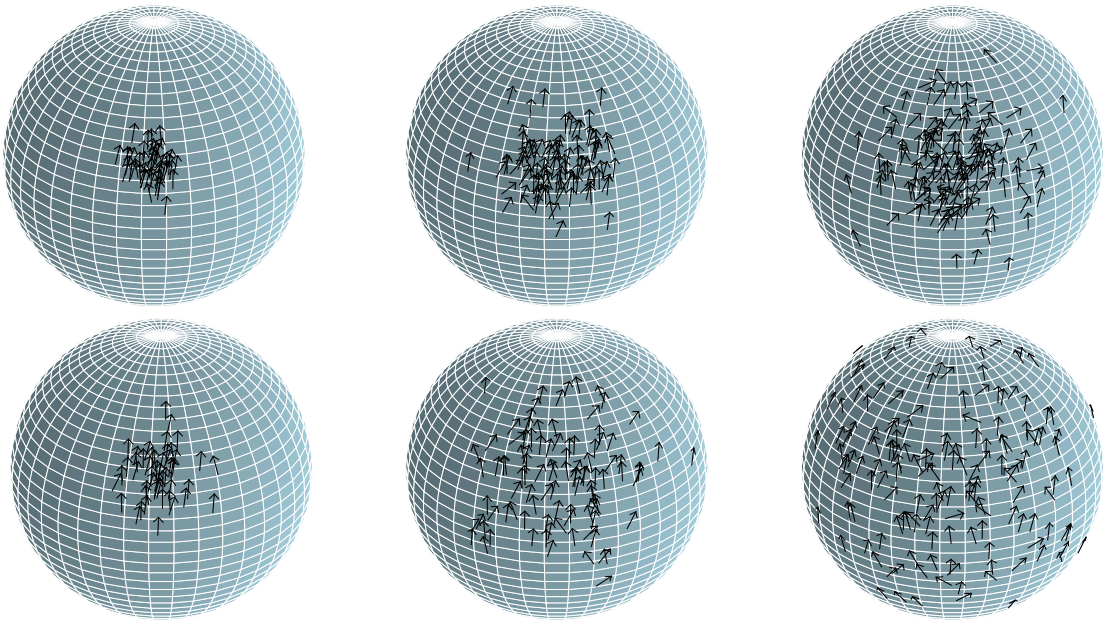


FIGURE 3.2: Probability distributions over rotations, visualized by transforming an arrow on the surface of a sphere according to sampled rotations: Defining distributions in the wrong tangent space can lead heavy to distortions. We use isotropic 3D normal distributions with three different standard deviations to sample points in different tangent spaces of $SO(3)$. **Top:** We define the distribution in the tangent space at the identity. The center of the distribution lies at $(0, \pi, 0)^\intercal$. We map the sampled points to $SO(3)$ via Eq. 3.9. This procedure leads to distorted distributions on $SO(3)$. **Bottom:** We define the distributions in the tangent space at the center of the distribution and map the sampled points to $SO(3)$ via Eq. 3.11. This procedure resolves the distortion. Note that we use the same standard deviations in the top and bottom row.

$p^b_{R_1}(\log_{R_1}(R))$ as one might expect because the two distributions are defined in different tangent spaces. They are thus essentially distributions of different random variables. Instead, if we want to do a meaningful comparison, we have to compare probability densities in the same tangent space. We propose to do the comparison of densities in the tangent space at R, *i.e.* at the very location we are interested in. [4]

We can use Eq. 3.13 to map elements from the tangent spaces at R_0 and R_1 to their representation in the tangent space at R. The elements sampled from $p^a_{R_0}(\boldsymbol{\omega}_{R_0})$ and $p^b_{R_1}(\boldsymbol{\omega}_{R_1})$ will follow a different distribution when mapped to their representation in the tangent space at R, where they should be compared. We will denote the probability density functions of these distributions as $p^b_R(\boldsymbol{\omega}_R)$ and $p^a_R(\boldsymbol{\omega}_R)$.

Remembering Eq. 3.12, we can see that the location R, we are interested in will be represented at the origin of the tangent space at R. Thus we have to compare the

[4] We choose to do this location purely for convenience. In principle, the comparison can be done in any tangent space: If we were to represent the location R in a different tangent space, the densities of both distributions would change by a common factor. The ratio of densities would thus remain unchanged.

probability densities $p^b_R(\boldsymbol{\omega}_R)$ and $p^a_R(\boldsymbol{\omega}_R)$ at the origin $\boldsymbol{\omega}_R = (0,0,0)^\intercal$. Our goal is now to calculate the ratio

$$\frac{p^a_R\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal\right)}{p^b_R\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal\right)}. \tag{3.17}$$

We can calculate the individual densities as

$$p^a_R\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal\right) = p^a_{R_0}\left(\boldsymbol{\omega}_{R_0} = \log_{R_0}(R)\right)\phi(\psi_{R_0,R}). \tag{3.18}$$

The calculation for $p^b_R\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal\right)$ can be done correspondingly. Based on the assumption $|\boldsymbol{\omega}_{R_0}| < \pi$, we can calculate the key factor as

$$\phi(\psi_{R_0,R}) = \frac{\psi^2_{R_0,R}}{2(1-\cos\psi_{R_0,R})}. \tag{3.19}$$

It is a function of the angle $\psi_{R_0,R}$ between the two tangent space locations R_0 and R as defined in Eq. 3.5. A derivation of $\phi(\psi_{R_0,R})$ can be found in section A.1. [5] We can now proceed to calculate the ratio of probability densities as

$$\frac{p^a_R\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal\right)}{p^b_R\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal\right)} = \frac{p^a_{R_0}\left(\boldsymbol{\omega}_{R_0} = \log_{R_0}(R)\right)\phi(\psi_{R_0,R})}{p^b_{R_1}\left(\boldsymbol{\omega}_{R_1} = \log_{R_1}(R)\right)\phi(\psi_{R_1,R})}. \tag{3.20}$$

In summary, we have discussed probability distributions in the tangent spaces of $SO(3)$ and how to compare them. In order to avoid distortions, it is sensible to define distributions in the tangent space at the distribution center. We can compare the probability densities of two distributions defined in different tangent spaces by multiplying a factor (see Eq. 3.20). In order for this method to be applicable, we require the probability densities to be restricted to the volume $|\boldsymbol{\omega}_{R_0}| < \pi$ and $|\boldsymbol{\omega}_{R_1}| < \pi$. In the following section, we will discuss a useful class of distributions that fulfills this requirement.

UARS distributions

We will now take a look at a particular class of probability distributions in $SO(3)$ termed uniform axis random spin (UARS) [91].

In [105] Qiu provides an introduction to the class and an overview of the many well studied distributions on $SO(3)$ that fall into this category. This includes the isotropic matrix version of the *von Mises-Fisher distribution* [108, 109], *Bunge's Gaussian distribution* [110], and the *isotropic normal distribution* on $SO(3)$ [111, 112] UARS distributions will play an important role in Chapter 4, where we will use them as part of the PF framework.

[5] A different derivation of the same factor is given by Vvedensky in [107].

We will introduce UARS distributions by describing the corresponding sampling process and then discuss how to define and compare their densities in tangent spaces in the sense of Eq. 3.20. Finally, we will see that this procedure is equivalent to using the density with respect to the Haar measure provided in [91].

UARS distributions describe a random perturbation around a specified rotation R_0. They take R_0 as staring point and apply a rotation of a randomly selected angle around a uniformly selected random axis.

They are defined through the following sampling process: First, a random, uniformly distributed rotation axis is drawn. This can be achieved by sampling a point from a 3D isotropic, zero centered Gaussian distribution and normalizing it to create a random unit vector $\hat{\boldsymbol{u}}$. Then, a rotation angle ψ is drawn from a circular distribution with density $C(\psi;\kappa)$ over the interval $(-\pi, pi]$. The parameter κ is a concentration parameter determining how far $C(\psi;\kappa)$ is spread out.

The distribution $C(\psi;\kappa)$ is required to be symmetric about 0. Depending on the particular choice of $C(\psi;\kappa)$, UARS can be shown to be equivalent to other classes of distributions. In [105] Qui gives a summary of these correspondences for different $C(\psi;\kappa)$.

The angle, sampled from $C(\psi;\kappa)$, and the axis $\hat{\boldsymbol{u}}$ are combined

$$\boldsymbol{\omega}_{R_0} = \psi\hat{\boldsymbol{u}} \tag{3.21}$$

and interpreted as a point in the tangent space at R_0. Finally, the point is mapped to $SO(3)$ via Eq. 3.11.

We can express UARS distributions as probability density in the tangent space at R_0:

$$p_{R_0}^{\text{UARS}}(\boldsymbol{\omega}_{R_0}; R_0, \kappa) = \frac{C(\psi = |\boldsymbol{\omega}_{R_0}|;\kappa)}{2\pi|\boldsymbol{\omega}_{R_0}|^2}. \tag{3.22}$$

A derivation of Eq. 3.22 can be found in the Appendix in Section A.2. Note that since $C(\psi;\kappa)$ is limited to $(-\pi, pi]$, Eq. 3.22 is limited to the domain where $|\boldsymbol{\omega}| < \pi$. This is exactly the requirement we made for the use of Eq. 3.20. We can thus compare the densities of two UARS distributions with parameters (κ_a, R_0) and (κ_b, R_1), at a particular rotation R by calculating:

$$\frac{p_R^{\text{UARS}}\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal; \kappa_a, R_0\right)}{p_R^{\text{UARS}}\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal; \kappa_b, R_1\right)} = \frac{p_{R_0}^{\text{UARS}}\left(\boldsymbol{\omega}_{R_0} = \log_{R_0}(R); \kappa_a, R_0\right)\phi(\psi_{R_0,R})}{p_{R_1}^{\text{UARS}}\left(\boldsymbol{\omega}_{R_1} = \log_{R_1}(R); \kappa_b, R_1\right)\phi(\psi_{R_1,R})}, \tag{3.23}$$

where the individual densities in the numerator and denominator can be expressed as

$$p_R^{\text{UARS}}\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal; \kappa_a, R_0\right) \tag{3.24}$$

$$= \; p_{R_0}^{\text{UARS}}\left(\boldsymbol{\omega}_{R_0} = \log_{R_0}(R); \kappa_a, R_0\right) \phi(\psi_{R_0,R}) \tag{3.25}$$

$$= \; \frac{C(\psi = |\boldsymbol{\omega}_{R_0}|; \kappa)}{4\pi\left(1 - \cos(|\boldsymbol{\omega}_{R_0}|)\right)} \tag{3.26}$$

The calculation for $p_R^{\text{UARS}}\left(\boldsymbol{\omega}_R = (0,0,0)^\intercal; \kappa_b, R_1\right)$ can be done correspondingly.

Note that Eq. 3.26 is proportional to the measure theoretic density with respect to the Haar measure, given in [91]:

$$f_{\text{UARS}}(R; R_0, \kappa) = \frac{4\pi}{3 - \text{tr}(R_0^{-1}R)} C\left(\arccos\left(\frac{1}{2}(\text{tr}(R_0^{-1}R) - 1)\right); \kappa\right). \tag{3.27}$$

The proportionality becomes clear by using $|\boldsymbol{\omega}_{R_0}| = \psi(R, R_0) = \arccos\left(\frac{1}{2}(\text{tr}(R_0^{-1}R) - 1)\right)$. It is thus equivalent to compare two UARS distributions through Eqs. 3.23 and 3.26 or to compare them by using the density with respect to the Haar measure given in Eq. 3.27.

In summary, the class of UARS distributions provides a useful tool, to model symmetric probability distributions over rotations. Since their probability density function in the tangent space is limited to $|\boldsymbol{\omega}_{R_0}| < \pi$ we can directly apply our mathematical tools from Section 3.1.4 and compare the probability densities in a meaningful way. The comparison is equivalent to using the densities with respect to the Haar measure as given in [91] (Eq. 3.27). We can apply UARS distributions in the context of importance sampling methods, or other procedures requiring comparison of probability densities.

3.2 A Probability Distribution over Poses

Let us finally consider the problem of defining a local parametric distribution over poses. We would like to define a distribution over poses $H = (R, \boldsymbol{v})$ that is, similarly to a normal distribution or a UARS distribution, concentrated around a center pose $H_0 = (R_0, \boldsymbol{v}_0)$. We will refer to this distribution as *UARS-Normal* distribution.

To achieve this, we will consider rotation R and translation $\boldsymbol{v}$ to be two independent random variables. We model the translational part as a 3D multivariate normal distribution $f_{\mathcal{N}}(\boldsymbol{v}; \Sigma, \boldsymbol{v}_0)$ over $\mathbb{R}^3$, centered at $\boldsymbol{v}_0$ and with covariance Σ. The rotational part is modeled as UARS distribution $f_{\text{UARS}}(R; R_0, \kappa)$. Even though UARS distributions can be defined based on any symmetric circular distribution $C(\psi; \kappa)$, we will assume ψ to follow a *von Mises distribution* (see [113]). The von Mises distribution has

a density function that can be feasibly evaluated, and it is a close approximation of the wrapped normal distribution [114].

Since rotation and translation are independent, we can sample from it by generating R and $\boldsymbol{v}$ separately. We define a probability density as the product

$$f_{\mathrm{UARS-N}}(H; H_0, \Sigma, \kappa) = f_{\mathcal{N}}(\boldsymbol{v}; \Sigma, \boldsymbol{v}_0) f_{\mathrm{UARS}}(R; R_0, \kappa). \tag{3.28}$$

We will refer to this kind of distribution as a *UARS-Normal distribution*. Even though this class of distribution is somewhat limited, as it assumes independence between rotation and translation, it will be an important building block in the systems described in the following Chapters 4 and 5.

Chapter 4

System I: Pose Tracking

Contents

In this chapter we use the one-shot pose estimation framework (see Section 2) as a starting point to develop a system for online pose tracking (see Section 1.1.2). The system uses sampling to represent and to reason with uncertainties. Note that the system has been published in [48].

4.1 Introduction

Many applications for object pose estimation require not a single estimated pose, but rather a stream of pose estimates produced in real-time from a stream of visual input: In AR systems [30] a new frame is rendered multiple times per second and might require updated pose estimates. An autonomous vehicle [27–29] can benefit from a constant flow of new pose estimates for itself and other traffic participants. A robotic system that attempts to grasp an object [26] handed to it by a human can readjust its movements by making use of real-time pose estimates.

One way to approach this problem is to simply run a fast one-shot pose estimation system repeatedly for every frame. There are two reasons this solution is sub-optimal:

First, the system would not make use of all available information, as it disregards everything observed in previous frames. This could lead to extreme error in individual difficult frames, showing e.g. a partially occluded object. Second, such a system would waste resources by starting a new search from scratch in every frame, even though the object is likely to be in the vicinity of the last estimated position.

We could mitigate these issues by modifying the system to only perform a local search centered around the last estimate. Such an approach would greatly reduce computation time and can even improve performance, as extreme outlier estimates become impossible. Indeed many successful tracking applications use this idea of a fixed search window or radius, e.g. [115, 116].

Note that the key ingredient to these improvements is an assumption we make about the possible movements of the object. We can only restrict ourself to the local search by assuming that the object cannot do arbitrary motions between two frames, but will stay in the vicinity of its last pose. Upon closer inspection, we find that there is an additional assumption at the core of this approach: In every new frame, we make the assumption that our last estimate was, at least approximately, correct. If this is not the case, we will search in the wrong area and are doomed to fail.

The tracking system we will present in this chapter is based on reconsidering the two assumptions by using probability theory. This idea is called *Bayes filtering* [117]. Instead of making the hard assumption that the object cannot move out of a certain window around the last pose, we use a probability distribution to describe the motion we expect between two frames. Instead of making the commitment that our last estimate was approximately correct, we use another probability distribution to describe our belief about the object's pose in the last frame.

When we observe a new frame, this latter probability distribution is updated to account for the newly observed information and for the fact that object might have moved in the mean time. There are different algorithms implementing Bayes filtering. The arguably most famous being the *Kalman filter* [118], which models said distributions as Gaussian. This approach however has some limitations, such as the fact that a Gaussian representation is uni-modal and quite restrictive. Since then, many extensions of the Kalman filter have been proposed to overcome these issues [119].

We will take a different path. The system we will present is based on a different implementation of the Bayes filter, namely the PF [32]. PF methods represent their belief about the current state by using a set of samples, referred to as *particles*. This is a much more flexible representation compared to the Gaussian one. In this chapter we will present a pose tracking method based on the online tracking formulation from 1.1.2. We will construct this system based on the PF recipe, but rely on many of the well-tried pose estimation ingredients from the one-shot framework by [31] (see Chapter 2).

One of the key ingredients for any PF-based system is its *observation model*. In our case, it describes the likelihood of observing a particular image, given the object is in a particular pose. We will use the scoring function from [31] in this context, as it can make use of the robust predictions made by the random forest.

The fact that the pose space is 6D, is potentially problematic for a PF system. The higher the dimensionality of the state space–in our case the 6D pose space–the more particles are required to adequately represent a probability distribution. To reduce the number of required particles one often uses a data-driven *proposal distribution* to distribute particles efficiently [117]. We make use of the hypotheses generation procedure from [31] to construct an effective proposal distribution. We evaluate our system on two datasets and are able to achieve superior results compared to a competitor and the one-shot system from [31].

4.1.1 Contributions

In summary, the main contributions presented in this chapter are:

- A new 6D pose tracking framework, based on the concept of predicting 3D object coordinates, which helps to generalize better to real-world settings of fast-moving objects and occlusions.
- A novel proposal distribution, based on object coordinate regression.
- A system that exceeds previous state-of-the-art results on 6D model-based tracking.

The rest of the chapter is structured as follows: We will start with a survey of the related work (Section 4.2) followed by a description of the system itself (Section 4.3). We will then continue to describe the experiments (Section 4.4) and end with a short conclusion (Section 4.5).

4.2 Related Work

Almost two decades ago the PF has been introduced for 2D visual tracking by Isard and Blake [120]. Based on a statistical observation model and a motion model the PF approximates the posterior distribution of the object's position in a non parametric form, using a set of samples. Ten years later Pupilli et al. [121] adapted the framework to 6-DOF camera tracking using edge features. Shortly later Klein et al. [101] presented an implementation that utilized the GPU for the evaluation of its observation model, which usually is the bottleneck in PF applications. The GPU implementation enabled them to deal with hidden edges while allowing a speedup [101].

The number of necessary particles and therefore the runtime can be reduced by guiding particle sampling with a proposal distribution. Ideally the proposal distribution is very close to the posterior distribution. Furthermore, it needs to exploit both the observation model and the motion model in order to improve over the standard PF as well as over one shot pose estimation. Bray et al. [122] improved hand pose tracking with a proposal distribution, which was defined as the mixture of two distributions both represented as a particle ensemble. The first particle ensemble is constructed in the default manner by applying the motion model to the sampled posterior distribution of the previous frame. The second ensemble is constructed by moving the resulting particles further to local optima and using them as centers of a mixture of normal distributions. Teuliere et al.[123] used a similar approach to edge-based tracking of simple objects from luminance images. Corresponding approaches for 3D pose tracking from RGB-D videos can be found in [37, 38, 124].

A PF that has to operate on the 6D Euclidean group SE(3) brings some theoretical challenges. The definition of probability distributions and calculation of average rotations is not straightforward. A theoretical analysis of these issues can be found in [125] and [104]. In this work we will consider the rotational and translational components of $SE(3)$ separately (see Chapter 3).

The question of the best image features for the observation model has sparked recent research. Stückler et al. [126] use RGB-D images to learn 3D surfel maps of objects and use them in a PF operating on RGB-D. Choi et al. [127] present in their recent work a highly optimized GPU implementation that uses a traditional mesh representation. Their observation model is based on comparing rendered images and observations using color as well as depth features.

In this work we build a PF tracking system using the scoring function of [31] (see Figure 1.4 g or Section 2.4). We carefully design a proposal distribution that intelligently exploits the input RGB-D image as well as the output of the trained random forest.

In this way we combine the robustness of the forest with respect to changes in illumination and the robustness of the PF framework with respect to occlusions and other visual ambiguities. Finally, our approach also uses GPU rendering to efficiently evaluate the observation model.

4.3 Method

The Method Section of this chapter is structured as follows: In Section 4.3.1 we will reiterate the online tracking task and describe the Bayes filtering framework for our setting. Then, in Section 4.3.2, we will describe our PF system. We will continue with

a description of our motion model (Section 4.3.3) and our definition of the observation likelihood (Section 4.3.4). Finally, in Section 4.3.5 we will describe how we adapt sampling of particles according to a proposal distribution which concentrates on areas where high particle likelihood is expected. This is a main component to facilitate efficient and robust pose tracking.

4.3.1 Bayes Filter

Our system addresses the online tracking problem as defined in Section 1.1.2. Given a stream of observations $\boldsymbol{z}_1 \dots \boldsymbol{z}_t$ of a moving object, our goal is to find an estimated object pose $\hat{H}_t = (\hat{R}_t, \hat{\boldsymbol{v}}_t)$ for each frame t, while making use of all previous information $\boldsymbol{z}_1 \dots \boldsymbol{z}_t$.

We assume our observations $\boldsymbol{z}_t$ to be the combination of a recorded RGB-D image $\boldsymbol{x}$ and the forest predictions, which are computed in a preprocessing step as described in Section 2.1.

To solve this task, our Bayes filter approach assumes an underlying probabilistic model called *Hidden Markov Model* (HMM) [128]. The HMM describes a probability distribution over a discrete time series of two variables:

- The states $(\mathcal{X}_1 \dots, \mathcal{X}_t)$. In our setting we define the state $\mathcal{X}_t$ as combination

$$\mathcal{X}_t = (H_t, \dot{\boldsymbol{v}}_t, \dot{\boldsymbol{\omega}}_t), \tag{4.1}$$

  of the pose H_t and two 3D velocity vectors $\dot{\boldsymbol{v}}_t$ and $\dot{\boldsymbol{\omega}}_t$, describing the translational and rotational motion of the object between the current and previous frame.

- The observations $\boldsymbol{z}_t$, as discussed above.

Based on the probabilistic framework of the HMM, we are mainly interested in the posterior distribution $p(\mathcal{X}_t|\boldsymbol{z}_{1:t})$ of the state $\mathcal{X}_t$ at time t conditioned on all previous observations $\boldsymbol{z}_{1:t} = \boldsymbol{z}_1 \dots \boldsymbol{z}_t$. This posterior corresponds to the system's belief about the current state. It will be the foundation of our pose estimates $\hat{H}_t$.

Bayes filters maintain a representation of the posterior and update it with every new observation. For this, they rely on the following property of the HMM: By combining the posterior distribution $p(\mathcal{X}_t|\boldsymbol{z}_{1:t})$ at time t with a new observation $\boldsymbol{z}_{t+1}$, we can recursively calculate the posterior $p(\mathcal{X}_{t+1}|\boldsymbol{z}_{1:t+1})$ for the new frame

$$\begin{aligned} p(\mathcal{X}_{t+1}|\boldsymbol{z}_{1:t+1}) &\propto p(\boldsymbol{z}_{t+1}|\mathcal{X}_{t+1}) p(\mathcal{X}_{t+1}|\boldsymbol{z}_{1:t}) && (4.2) \\ &= p(\boldsymbol{z}_{t+1}|\mathcal{X}_{t+1}) \int p(\mathcal{X}_{t+1}|\mathcal{X}_t) p(\mathcal{X}_t|\boldsymbol{z}_{1:t}) d\mathcal{X}_t, && (4.3) \end{aligned}$$

where $p(\boldsymbol{z}_{t+1}|\mathcal{X}_{t+1})$ is referred to as the *observation model* and $p(\mathcal{X}_{t+1}|\mathcal{X}_t)$ as the *motion model*.

The observation model describes the likelihood of an observation $\boldsymbol{z}_{t+1}$ given a state $\mathcal{X}_{t+1}$. Since our observations do not depend on the velocity of the object, we will define our observation model $p(\boldsymbol{z}_{t+1}|H_{t+1}) = p(\boldsymbol{z}_{t+1}|\mathcal{X}_{t+1})$ as purely depended on the pose. We will discuss our observation model in Section 4.3.4.

The motion model describes the probability of a state $\mathcal{X}_{t+1}$, given the previous state $\mathcal{X}_t$. It is a probabilistic model for the object motion we expect to see between two frames. Since the velocity vectors are calculated deterministically from the previous motion, we will describe only the probability $p(H_{t+1}|\mathcal{X}_t = (H_t, \dot{\boldsymbol{v}}_t, \dot{\boldsymbol{\omega}}_t))$ of the new pose instead of $p(\mathcal{X}_{t+1}|\mathcal{X}_t)$. We will discuss our motion model in Section 4.3.3.

4.3.2 PF for Pose Tracking

PFs approximate the posterior as a set of samples. Following this approach, we describe the current posterior distribution $p(\mathcal{X}_t|\boldsymbol{z}_{1:t})$ as a set of samples $P_t = \{H_t^1, \dots, H_t^K\}$, with velocity vectors $\dot{\boldsymbol{v}}_t^k$ and $\dot{\boldsymbol{\omega}}_t^k$ attached to each H_t^k. These samples will be referred to as particles. We use K to denote the number of particles, which is fixed over time.

With each new frame $t+1$ a new set of particles P_{t+1} is found in three steps:

1. **Sampling:** For each particle H_t^k an intermediate particle $\tilde{H}_{t+1}^k = \left(\tilde{\boldsymbol{v}}_{t+1}^k, \tilde{R}_{t+1}^k\right)$ is sampled according to the motion model $p(\tilde{H}_{t+1}^k|H_t^k, \dot{\boldsymbol{v}}_t^k, \dot{\boldsymbol{\omega}}_t^k)$. New velocities are calculated as

$$\dot{\boldsymbol{v}}_{t+1}^k = \tilde{\boldsymbol{v}}_{t+1}^k - \boldsymbol{v}_t^k \;\text{ and }\; \dot{\boldsymbol{\omega}}_{t+1}^k = \log\left(\tilde{R}_{t+1}^k R_t^{k^{-1}}\right), \tag{4.4}$$

   where $\log(\cdot)$ is the logarithmic map as defined in Section 3.1.2.

2. **Weighting:** Each intermediate particle is assigned a weight w_{t+1}^k, which is proportional to the likelihood $p(\boldsymbol{z}_{t+1}|\tilde{H}_{t+1}^k)$ of the observed data given the pose $\tilde{H}_{t+1}^k$.

3. **Resampling:** Finally, the set $P_{t+1} = \{H_{t+1}^1, \dots, H_{t+1}^K\}$ of unweighted particles (with their attached velocities), is randomly drawn from $\{\tilde{H}_{t+1}^1, \dots, \tilde{H}_{t+1}^K\}$ using probabilities proportional to the weights w_{t+1}^k.

After each new iteration of the PF we calculate our pose estimate $\hat{H}_t$ by separately averaging the translational and rotational (see Eq. 3.1.3) components of all particles H_t^k.

The number of particles required to approximate the 6D posterior distribution can be drastically reduced if the sampling is concentrated in areas where one expects the true pose of the object. This is done using a proposal distribution $q(H_{t+1}|H_t, \dot{\boldsymbol{v}}_t, \dot{\boldsymbol{\omega}}_t)$ in the sampling step instead of the original motion model. To compensate for the

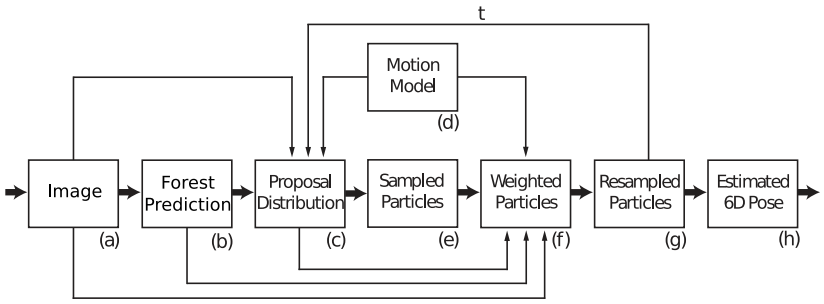


FIGURE 4.1: Our tracking pipeline. For each frame t the RGB-D image (**a**) is processed by the forest to predict object probabilities and local object coordinates (**b**). We use the observed depth from the original image, the forest predictions and the particles from the last frame together with our motion model (**d**) to construct our proposal distribution (**c**). Particles are sampled (**e**) according to the proposal distribution, then weighted (**f**) and resampled (**g**). Our final pose estimate is calculated as mean of the resampled particles (**h**).

fact that we sample from a different distribution, the calculation of weights has to be adjusted according to Eq. 19 in [129]:

$$w_{t+1}^k \propto p(\boldsymbol{z}_{t+1}|\tilde{H}_{t+1}^k)\frac{p(\tilde{H}_{t+1}^k|H_t^k, \dot{\boldsymbol{v}}_t^k, \dot{\boldsymbol{\omega}}_t^k)}{q(\tilde{H}_{t+1}^k|H_t^k, \dot{\boldsymbol{v}}_t^k, \dot{\boldsymbol{\omega}}_t^k)}. \tag{4.5}$$

This procedure of sampling from a proposal distribution and using the ratio of probability densities to compensate this, is known as importance sampling. Note that as opposed to the basic particle filter scheme described above, importance sampling requires the explicit calculation of probability densities in pose space (at least up to a common factor).

In the following we describe the specifics of our implementation of the PF framework: the motion model, the observation likelihood, and finally, our main contribution, the construction of our proposal distribution. An overview of our tracking pipeline can be found in Figure 4.1.

4.3.3 Our Motion Model

The motion model describes which movements of the object are plausible between two frames. Generally speaking, we assume our object roughly continues its previous motion, followed by an additional random normally distributed translation and a random rotation around the center of the object. We assume that the translational and rotational motion to be independent and describe them separately.

Given a translational component $\boldsymbol{v}_t$, we assume new translational component at the time point $t+1$ to be generated from a normal distribution

$$\boldsymbol{v}_{t+1} \sim \mathcal{N}\left(\boldsymbol{v}_t^{\text{pred}}, \Sigma^{\text{mm}}\right) \tag{4.6}$$

centered at the predicted translational component

$$\boldsymbol{v}_t^{\text{pred}} = \boldsymbol{v}_t + \lambda_{\boldsymbol{v}} \dot{\boldsymbol{v}}_t, \tag{4.7}$$

extrapolated from the last position $\boldsymbol{v}_t$ and the translational velocity $\dot{\boldsymbol{v}}_t$. The term $\lambda_{\boldsymbol{v}}$ stands for a damping parameter. It determines how much the previous translation, described by the velocity vector $\dot{\boldsymbol{v}}_t$ is continued. The covariance matrix is of the form $\Sigma^{\text{mm}} = I\sigma_{\boldsymbol{v}}^2$.

We assume a similar process for the generation of the new rotational component, but here special care has to be taken because of the internal structure of the Lie group $SO(3)$ (see Section 3.1).

Given the current rotational component R_t, we assume a new rotational component at the time point $t+1$ to be generated according to a UARS distribution (see Section 3.1.4), centered at the predicted rotational component that is extrapolated from the previous rotation R_t and the rotational velocity $\dot{\boldsymbol{\omega}}_t$:

$$R_t^{\text{pred}} = \exp_{R_t}(\lambda_R \dot{\boldsymbol{\omega}}_t), \tag{4.8}$$

where $\exp_{R_t}(\cdot)$ is the exponential map at R_t as defined in Eq. 3.11, and λ_R is again a damping parameter controlling how strongly the previous rotation is continued. The new rotational component is found according to the UARS procedure described in Section 3.1.4. A rotation angle $0 < \psi < \pi$ with standard deviation σ_R^{mm} is sampled from a wrapped normal distribution and combined with a uniformly chosen rotation axis to create an Euler vector $\boldsymbol{\omega}_{R_t^{\text{pred}}}$. The new rotation is then computed as

$$R_{t+1} = \exp_{R_t^{\text{pred}}}(\boldsymbol{\omega}_{R_t^{\text{pred}}}). \tag{4.9}$$

Based on the sampling procedure above, we can describe our motion model as a UARS-Normal distribution (see Section 3.2) with density

$$p(H_{t+1}|H_t, \dot{\boldsymbol{v}}_t, \dot{\boldsymbol{\omega}}_t) \approx f_{\text{UARS-N}}(H_{t+1}; H_t^{\text{pred}}, \Sigma^{\text{mm}}, \kappa^{\text{mm}}) \tag{4.10}$$

as defined in Eq. 3.28. [1]

This UARS-Normal distribution is centered at the predicted pose $H_t^{\text{pred}} = (R_t^{\text{pred}}, \boldsymbol{v}_t^{\text{pred}})$, which is found by extrapolating previous motion given by the velocity vectors $\dot{\boldsymbol{v}}_t$ and $\dot{\boldsymbol{\omega}}_t$ as described in Eqs. 4.8 and 4.7. The probability distribution is additionally parametrized on the variance in the translational component through $\Sigma^{\text{mm}} = I\sigma_{\boldsymbol{v}}^{\text{mm}2}$ and the variance of the rotational component through $\kappa^{\text{mm}} = 1/\sigma_R^{\text{mm}2}$.

[1] Note that Eq. 4.10 is only an approximate equality, since the UARS-Normal distribution uses a von Mises circular distribution, and our sampling process is defined on a wrapped normal distribution.

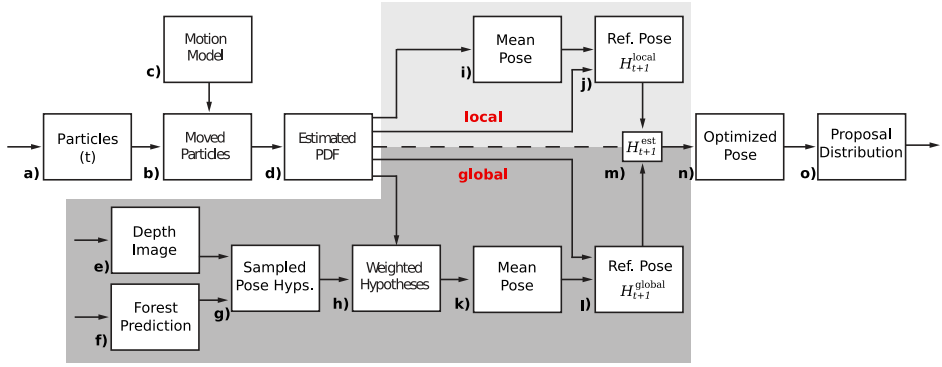


FIGURE 4.2: To construct our proposal distribution we first calculate a continuous representation of the prior distribution for the pose at the current frame (**a-d**). Next we determine two pose estimates H_{t+1}^{local} (light gray) and H_{t+1}^{global} (dark gray). The local estimate is calculated based on a local search on propagated solutions via the motion model (**k-j**). The global estimate is based on the pose sampling scheme from [31] (**e-l**). We choose the particle with the lower score and use it as starting point for a final optimization (**n**). Our final proposal distribution $q(H_{t+1}^k|H_t^k)$ for particle H_t^k is a mixture of two components (**o**): one centered on H_t^k and one on the newly found particle in (**n**). This figure represents component (c) of Figure 4.1.

4.3.4 Our Observation Likelihood

We use a likelihood formulation based on the scoring function $E(H)$ from [31] (see Section 2.4):

$$p(\boldsymbol{z}_{t+1}|H_{t+1}) \propto \exp(-\lambda_{\text{like}} E(H_{t+1})) \tag{4.11}$$

where λ_{like} is a control parameter determining how harshly larger values should be punished. The scoring function is a weighted sum of three components $E_{\text{depth}}(H)$, $E_{\text{coord}}(H)$ and $E_{\text{obj}}(H)$. In contrast to [31], we use a simple modification of the $E_{\text{depth}}(H)$ term that copes better with occlusion. Depth values that lie in front of the object can be explained by occlusion. This is not the case for depth values that lie behind the object. Our depth term accounts for this using a modified version of the depth error in Eq. 2.5. It uses a separate threshold τ_d for recorded depth values in front of the rendered object.

$$f_{\text{occ}}\left(\boldsymbol{d}_k, \tilde{\boldsymbol{d}}_k(H)\right) = \begin{cases} \min\left(||\boldsymbol{d}_k, \hat{\boldsymbol{d}}_k(H)||, \tau_d\right)/\tau_d & \text{if } \boldsymbol{d}_k < \hat{\boldsymbol{d}}_k(H) \\ \min\left(||\boldsymbol{d}_k, \hat{\boldsymbol{d}}_k(H)||, \tau_d^{\text{occ}}\right)/\tau_d & \text{else} \end{cases} \tag{4.12}$$

4.3.5 Our Proposal Distribution

Our proposal distribution allows our method to cope with unexpected motion and occlusion, while maintaining high accuracy. It allows us to approximate the posterior

distribution $p(H_t|\boldsymbol{z}_{1:t})$ more accurately with a small number of particles. The construction of our proposal distribution is described in the following, and subsumed in Figure 4.2.

A proposal distribution describes the sampling of a new particle $\tilde{H}^k_{t+1}$ on the basis of an old particle H^k_t. We define the proposal distribution $q(H_{t+1}|H_t, \dot{\boldsymbol{v}}_t, \dot{\boldsymbol{\omega}}_t)$ for the particle H^k_t as a mixture of two parts (Figure 4.2 o):

$$\begin{aligned} q(H^k_{t+1}|H^k_t, \dot{\boldsymbol{v}}^k_t, \dot{\boldsymbol{\omega}}^k_t) = (1-\alpha^{\text{prop}}) f_{\text{UARS-N}}(H^k_{t+1}; H^{k,\text{pred}}_t, \Sigma^{\text{prop}}, \kappa^{\text{prop}}) \\ +\alpha^{\text{prop}} f_{\text{UARS-N}}(H^k_{t+1}; H^{\text{est}}_{t+1}, \Sigma^{\text{prop}}, \kappa^{\text{prop}}) \end{aligned} \tag{4.13}$$

The mixture is governed by the weight α^{prop}. Both parts are based on UARS-Normal distributions defined in Eq. 3.28 with variance parameters Σ^{prop} and κ^{prop}. The first part of Eq. (4.13) is centered on $H^{k,\text{pred}}_t = (R^{k,\text{pred}}_t, \boldsymbol{v}^{k,\text{pred}}_t)$, which is the extrapolation of the current particle H^k_t according to our motion model as described in Eqs. 4.8 and 4.7. Hence, $H^{k,\text{pred}}_t$ differs for each particle H^k_t. The second part of Eq. (4.13) is centered on a preliminary estimate H^{est}_{t+1} which is found based on the output of the random forest (Section 2.1). It does not depend on H^k_t, but is shared among all particles.

Regarding H^{est}_{t+1}, we will discuss two different ways to quickly obtain a good estimate: One way finds a local estimate H^{local}_{t+1}, the other way finds a global estimate H^{global}_{t+1}. While a proposal distribution based on a local estimate is sufficient in most cases, it may fail in situations with fast unexpected motion. The global estimate on the other hand depends on the quality of the discriminative prediction and may at times give noisy results. As a consequence, we apply a combination of the two approaches:

$$H^{\text{est}}_{t+1} = \text{argmin}_{H\in\mathcal{H}} E'(H); \mathcal{H} = \{H^{\text{local}}, H^{\text{global}}\} \tag{4.14}$$

The scoring function $E'(H)$ will be defined below. The preliminary estimate H^{est}_{t+1} is optimized (Figure 4.2 n) using a general purpose optimizer. [2]

The remainder of this section is concerned with the calculation of H^{local}_{t+1} and H^{global}_{t+1}. First, however, we discuss how we represent prior knowledge used in both estimates.

Prior Knowledge

The proposal distribution should be an estimate of the posterior $p(H_{t+1}|\boldsymbol{z}_{1:t+1})$. Both of our estimates should thus include not only knowledge taken from the current observation k.e. the likelihood and results from the discriminative function, but also information from the previous particle set k.e. the prior. To include the prior we perform the following preparatory steps. We take the last set of particles $P_t = \{H^1_t, \ldots, H^K_t\}$

[2] We use the *Constrained Optimization BY Linear Approximations* (COBYLA) algorithm by Powell [130] in an implementation from the NLopt library [131].

and move each particle according to the motion model (Figure 4.2 a-c). The result is an extrapolated set of particles $\bar{P}_{t+1} = \{\bar{H}^1_{t+1}, \ldots, \bar{H}^K_{t+1}\}$. In order to obtain a parametric representation we again use a UARS-Normal distribution from Eq. 3.28. We fit $f_{\mathrm{UARS-N}}(H; H^{\mathrm{center}}, \Sigma, \kappa)$ to the particle set $\bar{P}_{t+1}$ (Figure 4.2 d). The resulting parameters are $\bar{H}^{\mathrm{center}}, \bar{\Sigma}, \bar{\kappa}$. For details on the fitting procedure, please refer to the Appendix (Section B.1). The distribution $f_{\mathrm{UARS-N}}(H; \bar{H}^{\mathrm{center}}, \bar{\Sigma}, \bar{\kappa})$ is a representation of the knowledge we have about the pose at the current time $t+1$ without considering the current observation $\boldsymbol{z}_{t+1}$. It is a representation of the prior.

Local Estimate

To find H^{local}_{t+1}, we use $\bar{H}^{\mathrm{center}}$ (Figure 4.2 k) as starting point for refinement as described in [31] (Figure 4.2 j). This refinement is done by repeatedly finding inlier pixels. Their predicted object coordinates together with the observed depth values enable a rough but quick optimization using the Kabsch algorithm. In order to include prior knowledge in the refinement we change the objective function to:

$$E'(H) = \lambda_{\mathrm{like}} E(H) - \ln f_{\mathrm{UARS-N}}(H; \bar{H}^{\mathrm{center}}, \bar{\Sigma}, \bar{\kappa}) \tag{4.15}$$

Because of this adjustment of the objective function the resulting H^{local}_{t+1} becomes a local maximum a posteriori (MAP) estimate.

Global Estimate

Calculation of the global estimate $H^{\mathrm{global}}_{t+1}$ is based on a sampling scheme similar to the one in [31]. We sample a set of m hypotheses $\check{H}^k$ (Figure 4.2 e-g). Details can be found in the Appendix (Section B.2). Then, the hypotheses $\check{H}^k$ are weighted using the distribution $f_{\mathrm{UARS-N}}(H; \bar{H}^{\mathrm{center}}, \bar{\Sigma}, \bar{\kappa})$ (Figure 4.2 h). Finally, their weighted mean is calculated (Figure 4.2 k) and used as initialization for the refinement (Figure 4.2 l), again using $E'(H)$ from Eq. (4.15) as objective function. This yields $H^{\mathrm{global}}_{t+1}$.

4.4 Experiments

Some RGB-D object tracking datasets have been published in recent years. For example, Fanelli et al. [132] recorded a dataset to track human head poses using a Kinect camera. Song and Xiao [133] used a Kinect camera to record 100 RGB-D video sequences of arbitrary objects but do only provide 2D bounding boxes as ground truth. For our purpose, we found only one relevant dataset from Choi and Christensen [127]. It consists of 3D object models and synthetic test sequences. For further evaluation, we recorded a new more challenging and realistic dataset on which we compared our

approach. Additionally, we conduct experiments to demonstrate that our proposal distribution achieves superior results when unexpected object motion occurs.

Note that the experiments described in this section, were conducted using an approximate UARS density function, as described in the Appendix in Section B.4.

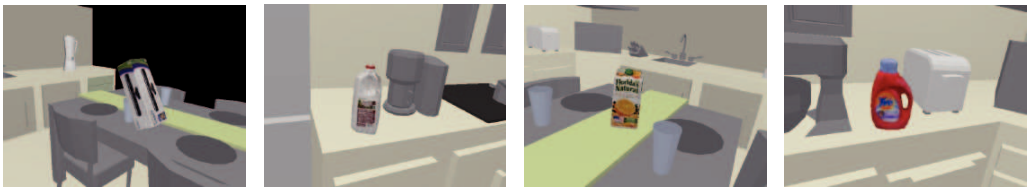


FIGURE 4.3: Example images of the dataset provided by Choi and Christensen [127].

Dataset of Choi and Christensen [127]

The dataset of Choi and Christensen [127] provides four textured 3D models and four synthetic test sequences (1000 RGB-D frames). To generate the test sequences, each of the four objects was placed in a static texture-less 3D scene and the camera was slowly moved around the object. The authors provide the ground truth camera trajectory which is error-free since it was generated through rendering. Figure 4.3 shows one image of each sequence.

To gather the training data for our random forest we rendered RGB-D images of each model. We sampled the full view sphere of the model regularly with fixed distance and including in-plane rotation. For the background set we used renderings from multiple 3D scenes from Google warehouse. We trained three decision trees with a maximum feature patch size of 20 pixel meter and 125 discrete labels per object. We trained the trees for all 4 objects jointly. For each testing sequence the object to be tracked is assumed to be known, and predictions are only made for this object. Our PF uses 70 particles. The complete list of parameters is included in the Appendix in Section B.3.

While testing we follow the evaluation protocol of Choi and Christensen [127] and compute the Root Mean Square Error (RMSE) of the translation parameters X,Y and Z and the rotation parameters Roll, Pitch and Yaw. We average the translational RMSE over three test runs, the coordinates (X,Y and Z), as well as over the four objects to obtain one translational error measure. We do the same for rotational RMSE. We compare to the numbers provided in [127] which also include results for the tracking implementation of the Point Cloud Library (PCL) [134]. We base our comparison on the results in [127] achieved with 12800 particles, for which the lowest error is reported.

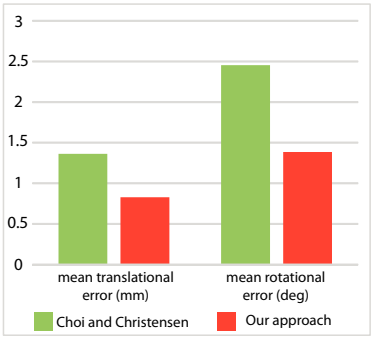


FIGURE 4.4: Averaged translation and rotation RMSE on the dataset of [127].

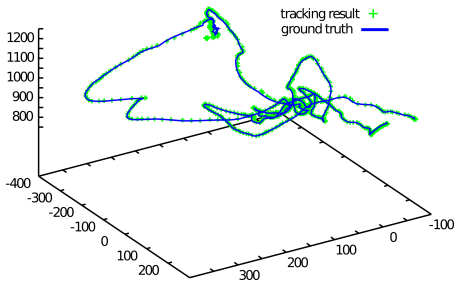


FIGURE 4.5: Reconstructed motion trajectory (green) for one sequence of our dataset (Cat, sequence 1). Ground truth is depicted blue for comparison. The positions are given in mm.

Our method results in an average translational RMSE of 0.83 mm compared to 1.36 mm for [127], k.e. we achieve a 38% lower translational error (PCL: 18.7 mm). For the average rotational RMSE we report 1.38 deg compared to 2.45 deg in [127], which is 43% lower (PCL: 29.6 deg). We achieve these results while keeping the computation time on our system[3] comparable to the one reported in [127]. Figure 4.4 depicts the average RMSE over all objects. Detailed results including run-times can be found in Table 4.1.

Objects	Tracker	RMSE						
		X (mm)	Y (mm)	Z (mm)	Roll (deg)	Pitch (deg)	Yaw (deg)	Time (ms)
Kinect Box	PCL	43.99	42.51	55.89	7.62	1.87	8.31	4539
	Choi and Christensen	1.84	2.23	1.36	6.41	0.76	6.32	166
	Our	**0.83**	**1.67**	**0.79**	**1.11**	**0.55**	**1.04**	**143**
Milk	PCL	13.38	31.45	26.09	59.37	19.58	75.03	2205
	Choi and Christensen	0.93	1.94	1.09	3.83	**1.41**	3.26	**134**
	Our	**0.51**	**1.27**	**0.62**	**2.19**	1.44	**1.90**	135
Orange Juice	PCL	2.53	2.20	1.91	85.81	42.12	46.37	1637
	Choi and Christensen	0.96	1.44	1.17	1.32	**0.75**	1.39	**117**
	Our	**0.52**	**0.74**	**0.63**	**1.28**	1.08	**1.20**	129
Tide	PCL	1.46	2.25	0.92	5.15	2.13	2.98	2762
	Choi and Christensen	0.83	1.37	1.20	**1.78**	**1.09**	**1.13**	**111**
	Our	**0.69**	**0.81**	**0.81**	2.10	1.38	1.27	116

TABLE 4.1: Comparison of the translation error (X,Y,Z), rotation error (Roll, Pitch, Yaw) and computation time on the synthetic dataset of Choi and Christensen [127] with results from our method, [127] and the PCL.

Our dataset

The dataset which was provided by [127] is problematic for several reasons: testing sequences are generated synthetically without camera noise, and without occlusion. The objects are placed in a texture-less and static environment. In a static scene, a

[3]Intel Core k7-3820 CPU @ 3.6GHz with a Nvidia GTX 550 TI GPU

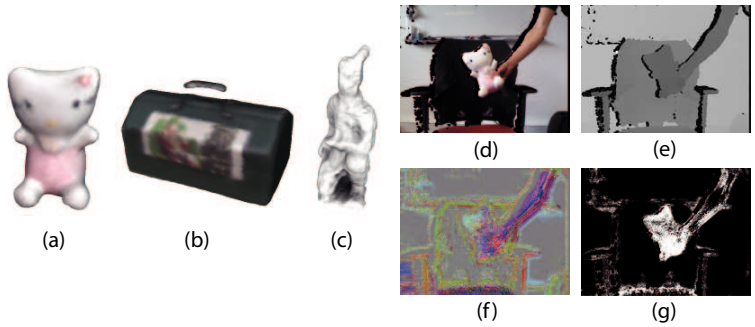


FIGURE 4.6: (a)-(c) Our objects from left to right Cat, Toolbox, Samurai. (d) color frame and (e) depth frame recorded with the commercially available Kinect camera.(f) Probability map and (g) predicted 3D object coordinates from a single tree mapped to the RGB-cube for the object Cat.

tracking method can in theory use the entire image to estimate the motion of the camera instead of estimating the motion of the object. Furthermore, the camera is moved around the object. The statistics of object motion when the camera is moved are very different from a situation where the camera is static and the object is moved. e.g., a complete vertical flip of the object is unlikely in the first scenario.

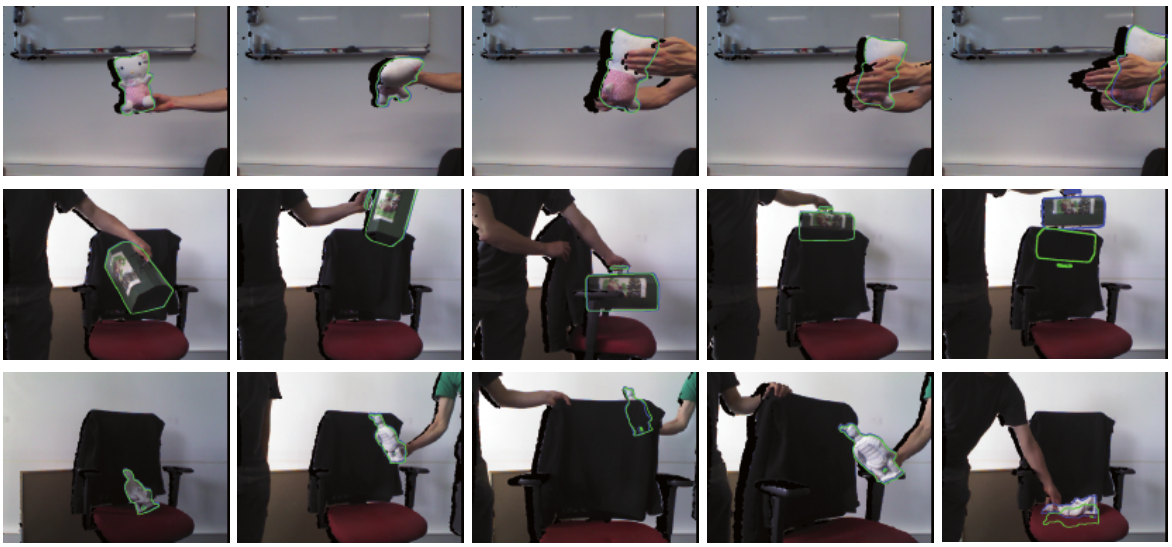


FIGURE 4.7: Example images from our dataset. Blue object silhouettes depict ground truth and green silhouettes depict the estimated poses. The first four columns show correctly estimated poses and the last column missed poses.

To address these issues we introduce our own dataset, which will be referred to as *the dataset by Krull et al.*. in the remainder of this thesis. The dataset consists of three objects. The objects were scanned in using Kinect, and six RGB-D testing sequences were recorded (350+ frames each). The objects are moved quickly in front of a static

camera and are at times strongly occluded. Ground truth poses were manually labeled by hand annotation followed by ICP. In Figure 4.7 five images of each object are shown. For our dataset we trained decision trees as discussed in the previous section, but with renderings of our scanned-in objects, and a set of arbitrary RGB-D office images as background set. We keep all other training parameters as in the previously described experiment. We compare our approach to the one shot pose estimation

Objects	Sequence	Ratio of frames used	Method		
			Our Approach		Brachmann et al.
			full proposal distribution	local proposal distribution	
			Accuracy	Accuracy	Accuracy
Cat	1	100%	**100%**	89%	90.5%
		33%	**91.5%**	90.4%	
	2	100%	99.4%	**100%**	44%
		33%	**94.9%**	87.8%	
Samurai	1	100%	96.3%	**96.7%**	71.29%
		33%	**68.6%**	52.4%	
	2	100%	92.3%	**98.1%**	35.5%
		33%	**55.4%**	29.1%	
Toolbox	1	100%	**88.8%**	88.5%	55%
		33%	**81.2%**	68.2%	
	2	100%	**100%**	**100%**	60.9%
		33 %	**89.9%**	34.5%	

TABLE 4.2: Accuracy measured on our dataset. Comparison of our full proposal distribution to the local proposal distribution and to [31]. Evaluation is done based on all frames and on every third frame of the image sequences.

from [31]. We exactly adhere to their training setup and parameters (3 trees, maximum patch size 20 pixel meter, 210 hypotheses per frame, Gaussian noise on training data). We measure accuracy as in [31] as the fraction of frames where the object pose was estimated correctly.

While our approach achieves 96.2% accuracy on average over all sequences the approach of [31] only estimates 59.5% [4] of the poses correctly. Even though [31] is inherently robust to occlusion, the heavy occlusions in our dataset still cause it to fail. In contrast, our approach is able to estimate most poses correctly by using information from previous frames. The distribution of errors for one sequence is depicted in Figure 4.8. The plots again show the large number of outlier estimations of [31] (rightmost bins). The plots also reveal that concerning correct poses, our approach leads to much more precise estimations.

[4] We re-ran the baseline of [31] for this dataset, as there was a typo in one of the originally published results. The re-run gave the average accuracy of 59.5%. The original number published in [48] was 58.9%. The results of the re-run are reported in Table 4.2.

To show that our full proposal distribution (Section 4.3.5) increases the robustness of our method we conducted the following experiment: We define a simplified variant of the proposal distribution, which is based only on H_{t+1}^{local} to which we also apply the final optimization. We term this variant *local proposal distribution*. We use it together with 120 particles since it needs less computation time. For this experiment,

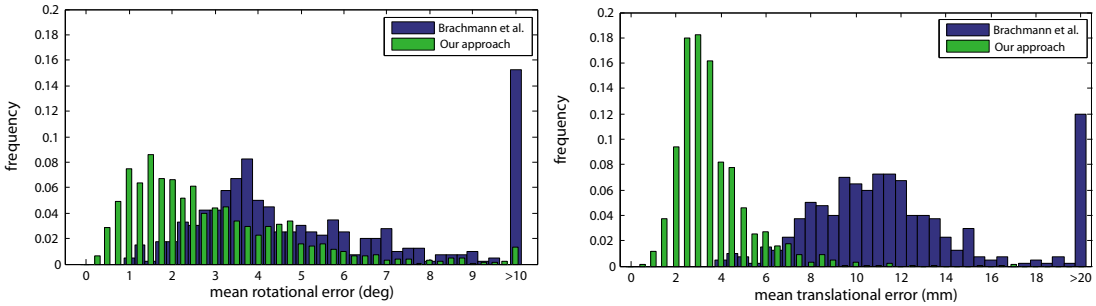


FIGURE 4.8: Histogram of rotational and translational errors for our approach in comparison to [31], which is a single frame pose estimation framework.

we artificially increase motion in our test sequences by using only every third frame. As before, we measure the number of correctly estimated poses. In this challenging setup the full proposal distribution achieves 80.3% accuracy on average while the local proposal distribution achieves only 60.4% accuracy.

Figure 4.5 shows the estimated object motion path for one sequence with fast motion. The plot illustrates precision and robustness of our approach.

4.5 Conclusion

We have introduced a novel method, applying the concept of predicted object coordinates to the problem of 6D pose tracking. We have extended the framework from [31] to allow probabilistic temporal reasoning in a HMM formulation. We utilize predicted object coordinates in a proposal distribution, making our method very robust with regard to fast movements in combination with occlusion. We have evaluated our method on the dataset by Coi and Christensen and demonstrated that it can produce superior results. The method was additionally evaluated using a new dataset, specially designed for RGB-D 6D pose tracking. The dataset has been made available to the community.

Chapter 5

System II: Learning Analysis-By-Synthesis

Contents

In this chapter, we will leave the tracking task behind, and focus again on the one-shot pose estimation task, as described in Section 1.1.1. We will present a CNN-based system that learns to assess the quality of pose hypothesis, by comparing rendered and observed images. The CNN will replace the simple pixel-wise distance function (see Section 2.4) used in the framework from [31] and greatly improve the robustness against occlusion. We use a sampling-driven probabilistic approach to train our system. Note that the system has been published in [81].

5.1 Introduction

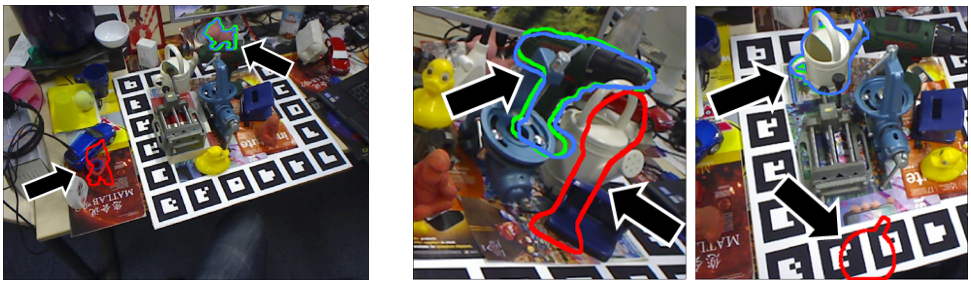


FIGURE 5.1: Three pose estimation results from the *occlusion dataset* from [31] and [14]. Arrows indicate the positions of estimated and ground truth poses. The green silhouette indicates the ground truth pose, the blue silhouette corresponds to our estimated pose. Red indicates the pose estimate from [31]. The marker board served only for ground truth annotation.

Analysis-by-synthesis has been a successful approach for many tasks in computer vision, such as object recognition [135], scene parsing [136], pose estimation, and tracking [137]. The basic idea is the following: First, a forward synthesis model generates images from a set of hypotheses. Then, one finds an estimate by selecting the hypothesis leading to the highest level of agreement with the measured visual evidence.

The framework from [31] implements this idea for the problem of object pose estimation, by rendering the object under different pose hypotheses and comparing the results to observed images. While the comparison for depth images is normally quite robust, comparing observed and rendered color images is more challenging, as the appearance of an object under identical pose can differ significantly due to light conditions (see Section 1.2). In [31] this problem is alleviated by giving up this comparison of the color channels, and instead comparing rendered and predicted object coordinates, which are more robust to lighting changes.

The situation becomes more difficult as soon the object is partially occluded. We see several reasons for this: (i) The simple pixel-wise comparison of depth channels can fail for occluded objects. Since an object can be occluded in many different ways and thus can appear very differently (see Section 1.2), the prediction of object coordinates in the presence of occlusion is difficult as well. (ii) Falsely predicted object coordinates resulting from occlusion are a second source of error in the comparison of images. (iii) Additional problems result from faulty or missing depth information. A depth sensor can fail to produce measurements due to multiple reasons, such as poor infra red reflectance of the surface. Due to the internal mechanism of sensors like *kinect* depth information is also often not provided for areas near an occlusion. While

the dense predictions provide [31] with some inherent stability with respect to occlusion, the framework from [31] struggles when used with occluded objects, due to the points mentioned above.

In this chapter, we will present a trainable system to mitigate these problems. We will replace the simple pixel-wise scoring function in [31] with a CNN-based *energy function* that learns to compare rendered and observed images. We view the CNN as part of an energy-based probability distribution. Training the CNN requires an integral of the pose space, which we approximate via *Markov Chain Monte Carlo* (MCMC) sampling.

We empirically observe that the CNN does not specialize on the geometry or appearance of specific objects. It can be used with objects of vastly different shapes and appearances, and in different backgrounds. We demonstrate a significant improvement compared to [31] on two different datasets which include a total of eleven objects, cluttered background, and heavy occlusion.

5.1.1 Contributions

We present the following major contributions in this chapter:

- We achieve considerable improvements over previous state-of-the-art methods of pose estimation in RGB-D images with heavy occlusion.
- To the best of our knowledge, this work is the first to utilize a CNN as a probabilistic model to learn to compare rendered and observed images.
- We observe that the CNN does not specialize to the geometry or appearance of specific objects, and it can be used with objects of vastly different shapes and appearances, and in different backgrounds.

The remainder of this chapter is organized as follows. Section. 5.2 provides an overview of related work. Our proposed approach is described in Section 5.3. In Section. 5.4, we evaluate our method on two datasets and compare it to the framework from [31], which was the state-of-the-art competitor. We will conclude the chapter in Section. 5.5.

5.2 Related Work

A large body of work in computer vision has focused on the problem of object detection and pose estimation. We provide a survey of the different approaches in Section 1.3. Here, we will focus on techniques that specifically address CNNs and analysis-by-synthesis.

5.2.1 CNNs

CNNs have been driving advances in computer vision in areas such as image classification [138], detection [139], recognition [140, 141], semantic segmentation [142], and human pose estimation [143]. The success of CNNs is attributed to their ability to learn rich feature representations as opposed to hand-designed features used in previous image classification methods.

In [144], rich image and depth feature representations have been learned with CNNs to detect objects in RGB-D images. In [145], CNNs are used to generate an RGB image given the set of 3D chair models, the chair type, viewpoint and color.

CNNs have been a part of complex pipelines. Recently Gupta *et al.* [146] presented work that uses object instance segmentation output from [144] to infer the 3D object pose in RGB-D images. Another CNN is used to predict the coarse pose of the object. It is trained using pixel normals in images containing rendered synthetic objects. This coarse pose is used to align a small number of prototypical models to the data, and place the model that fits the best into the scene. In contrast to the above approaches, we use a CNN to compare rendered and observed images and assess the quality of a pose. The output of our CNN is a single energy value, while in [146] the output of the CNN is the object pose.

In [147] Chopra *et al.* use a CNN to compare images and output a score. They learn, a similarity metric. While their learning process uses a discriminative loss function, ours is probabilistically motivated.

Chopra *et al.* use a *siamese* architecture is to map two faces to a feature space for comparison. Similarly, in [63] Wohlhart and Lepetit train a CNN to map image patches to a descriptor space, where pose estimation and object recognition is solved using the nearest neighbor method. In contrast, we feed the images jointly into the CNN and do a direct comparison.

Zbontar and LeCun [148] train a CNN to predict how well two image patches match and use it to compute the stereo matching cost. The cost is minimized by cross-based cost aggregation and semi-global matching, followed by a left-right consistency check to eliminate errors in occluded regions. While in [148] the CNN is used for comparing two image patches, our CNN is used to compare rendered and observed images.

5.2.2 Analysis-by-synthesis

Analysis-by-synthesis has been a successful approach for many tasks in computer vision, such as object recognition [135], scene parsing [136], viewpoint synthesis [135], material classification [149], and gaze estimation [150]. All these approaches use a

forward model to synthesize some form of image, which is then compared to observations.

Many works learn a feature representation and compare in feature space. For instance, in [135] the analysis-by-synthesis strategy has been used for recognizing and reconstructing 3D objects in images. The forward model synthesizes visual templates defined on invariant features. Gall *et al.* [137] propose an analysis-by-synthesis framework for motion capture and tracking. It combines patch-based and region-based matching to track body parts. Patch-based matching extracts correspondences between two successive frames for prediction as well as between the current image and a synthesized image to avoid drift.

The framework by Brachmann *et al.* [31] (see Chapter 2) that our system is based on, utilizes analysis-by-synthesis for 6D object pose estimation and achieved state-of-the-art results. However, for the problem of 6D pose estimation, due to occlusion or complicated sensor noise, it can be difficult to compare the observation with the output of a rendered image of the object of interest in a particular pose.

In this chapter, we propose an improvement, which draws on recent successes of CNNs. Different from aforementioned approaches, we model the posterior density of the object pose with a CNN that compares rendered and observed images. The network is trained with the maximum likelihood paradigm.

One of the most closely related works is [151]. They use a CNN as a part of probabilistic model. Their CNN is fed in a sequential manner, first with the rendered image, then with the observed image. This produces two feature vectors, which are compared in the subsequent step, to give the probability of the observed image. Another major difference is that our CNN is trained, while they use a pre-trained CNN as feature extractor.

5.3 Method

We will first reiterate the pose estimation task. Then, we will describe our probabilistic model. The heart of this model is a CNN, which will be discussed subsequently. This is followed by a description of our maximum likelihood training procedure of the probabilistic model. Finally, our inference procedure at test time is described. Fig. 5.2 gives an overview of our energy evaluation pipeline.

5.3.1 The Pose Estimation Task

In this system we address the one-shot pose estimation task as described in Section 1.1.1: Given an observation z we want to find an estimate $\hat{H}$ of the true pose H^*. We take

an RGB-D image $\boldsymbol{x}$ as input. As in the previous chapter, the term observation $\boldsymbol{z}$ or observed images will refer to two parts: (i) the forest predictions as described in [31], as well as (ii) the recorded depth image $\boldsymbol{d}$. The reason for this simplified view is that the focus of the chapter lies on the modeling of the posterior density and not on aspects of the random forest prediction.

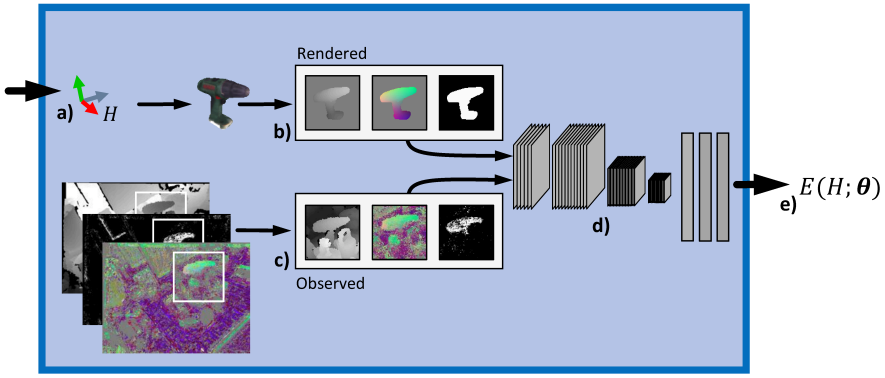


FIGURE 5.2: Our pipeline for the calculation of the energy function $E(H;\boldsymbol{\theta})$: **a):** We use the pose H to render the 3D model. **b):** We produce three rendered images: a rendered depth image, rendered image of object coordinates, and rendered mask. **c):** We crop the accordant image patches from the observed depth, object coordinate and object probability images. **d):** The images are processed by a CNN. **e):** The CNN has a single scalar output. It is interpreted as the energy $E(H;\boldsymbol{\theta})$.

5.3.2 Probabilistic Model

We model the posterior distribution of the pose H given the observations $\boldsymbol{z}$ as an energy-based distribution

$$p(H|\boldsymbol{z};\boldsymbol{\theta}) = \frac{\exp\big(-E(H,\boldsymbol{z};\boldsymbol{\theta})\big)}{\int \exp\big(-E(H',\boldsymbol{z};\boldsymbol{\theta})\big)dH'}\,, \tag{5.1}$$

where $E\left(H,\boldsymbol{z};\boldsymbol{\theta}\right)$ is the so-called energy function. The energy function is a mapping from a pose H and the observed images $\boldsymbol{z}$ to a real number. It is parametrized by the vector $\boldsymbol{\theta}$. Note that using a such a formulation to model the posterior is a common practice for conditional random fields (CRFs) [152]. However, the underlying energies are quite different. While in a CRF the energy function is a sum of potential functions, we implement it by using a CNN which directly outputs the energy value. The parameter vector $\boldsymbol{\theta}$ holds the weights of our CNN.

5.3.3 Convolutional Neural Network

In order to implement the mapping from a pose H and the observed images $\boldsymbol{z}$ to an energy value we first render the object in pose H to obtain rendered images $\acute{\boldsymbol{z}}(H)$. Our CNN then compares $\boldsymbol{z}$ with $\acute{\boldsymbol{z}}(H)$ and outputs a value $f\big(\boldsymbol{z}, \acute{\boldsymbol{z}}(H); \boldsymbol{\theta}\big)$. We define the energy function as

$$E(H, \boldsymbol{z}; \boldsymbol{\theta}) = f\big(\boldsymbol{z}, \acute{\boldsymbol{z}}(H); \boldsymbol{\theta}\big). \tag{5.2}$$

Our network is trained to assign a low energy value when there is a large agreement between observed images and renderings and a high energy value when there is little agreement. To perform the comparison we use a simple architecture, in which we feed all rendered and observed images jointly into the CNN.

Note that we consider only a square window around the center of the object with pose H. The width of the window is adjusted according to the size and distance of the object, as suggested by [31]. For performance reasons windows which are bigger than 100×100 pixels are down sampled to this size. We use a total of six input channels for our network. Note that Fig. 5.2 shows the images from which these six input channels are derived. The channels are:

- One **observed depth** channel and one **rendered depth** channel that contain values in millimeters. They are normalized by subtracting the z component of the object position according to H.
- One **rendered mask** channel of the object. Pixel values are either $+1$ for all pixels belonging to the object or -1 otherwise.
- One **depth mask** channel indicating whether a depth value was measured in the pixel. Again, pixel values are either $+1$ for all pixels where a depth was measured or -1 otherwise.
- One **probability** channel holding the combined pixel-wise object probabilities from all trees. The values are re-scaled to lie between -1 and $+1$.
- One **object coordinate** channel holding the pixel-wise Euclidean distances between the rendered object coordinates and the predicted object coordinates from the tree, giving the highest object probability for the respective pixel. We divide all values by the object diameter for normalization.

The $\tanh$ activation function is used after every convolution layer and after every fully connected layer. The first convolution layer C_1 consists 128 convolution kernels of size $3 \times 3 \times 6$. The second convolution layer C_2 consists of 128 kernels of size $3 \times 3 \times 128$, which is followed by a 2×2 max-pooling layer with stride 2 in each direction. The third convolution layer C_3 is identical to C_2. The fourth convolution layer

C_4 consists of 256 kernels of size $3 \times 3 \times 128$. It is followed by a max-pooling operation over the remaining image size. The 256 channels are further processed by two fully connected layers with 256 neurons each and finally forwarded to a single output unit.

5.3.4 Maximum Likelihood Training

During training, we want to find an optimal set of parameters $\boldsymbol{\theta}^*$ based on labeled training data $L = (\boldsymbol{z}_1, H_1^*) \dots (\boldsymbol{z}_n, H_n^*)$, where $\boldsymbol{z}_l$ shall denote observations of the l-th training image and H_l^* the corresponding ground truth pose. We apply the maximum likelihood paradigm and define

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{l=1}^{n} \ln p(H_l^* | \boldsymbol{z}_l; \boldsymbol{\theta}). \tag{5.3}$$

In order to solve this optimization task we use stochastic gradient descent [153], which requires calculating the partial derivatives of the log likelihood for each training sample

$$\frac{\partial}{\partial \theta_j} \ln p(H_l^* | \boldsymbol{z}_l; \boldsymbol{\theta}) = -\frac{\partial}{\partial \theta_j} E\left(H_l^*, \boldsymbol{z}_l; \boldsymbol{\theta}\right) + \mathbb{E}_{p(H|\boldsymbol{z}_l;\boldsymbol{\theta})} \left[\frac{\partial}{\partial \theta_j} E\left(H, \boldsymbol{z}_l; \boldsymbol{\theta}\right) \right] \tag{5.4}$$

with respect to each parameter θ_j. Here $\mathbb{E}_{p(H|\boldsymbol{z}_l;\boldsymbol{\theta})}[\cdot]$ stands for the expected value according to the posterior distribution $p(H|\boldsymbol{z}_l; \boldsymbol{\theta})$, parametrized by $\boldsymbol{\theta}$. While the partial derivatives of the energy function can be calculated by applying back propagation in our CNN, the expected value cannot be found in closed form. Therefore, we use a sampling approach to approximate it, as will be discussed next.

Sampling

The Monte Carlo method [2] allows us to approximate the expected value in Eq. (5.4) through a set of pose samples

$$\mathbb{E}_{p(H|\boldsymbol{z}_l;\boldsymbol{\theta})} \left[\frac{\partial}{\partial \theta_j} E\left(H, \boldsymbol{z}_l; \boldsymbol{\theta}\right) \right] \approx \frac{1}{K} \sum_{k=1}^{K} \frac{\partial}{\partial \theta_j} E\left(H_k, \boldsymbol{z}_l; \boldsymbol{\theta}\right), \tag{5.5}$$

where $H_1 \dots H_N$ are pose-samples drawn independently from the posterior $p(H|\boldsymbol{z}_l; \boldsymbol{\theta})$ with the current parameters $\boldsymbol{\theta}$.

However, the posterior defined in Eq. 5.1 cannot be easily evaluated, not to mention sampled from. We use an MCMC method, namely the Metropolis algorithm [3] to

generate the required samples. The Metropolis algorithm allows sampling from distributions with a density function that can be evaluated up to a constant factor. The algorithm generates a sequence of samples H_k by repeating two steps:

1. Draw a new proposed sample H' according to a proposal distribution $q(H'|H_k)$.
2. Accept or reject the proposed sample according to an acceptance probability $A(H'|H_k)$. If the proposed sample is accepted set $H_{k+1} = H'$. If it is rejected set $H_{k+1} = H_k$.

The proposal distribution $q(H'|H_k)$ has to be symmetric, *i.e.* $q(H'|H_k) = q(H_k|H')$. Our particular proposal distribution will be described in detail in the next paragraph. The acceptance probability is in our case defined as

$$A(H'|H_t) = \min\left(1, \frac{p(H'|\boldsymbol{z}_l;\boldsymbol{\theta})}{p(H_k|\boldsymbol{z}_l;\boldsymbol{\theta})}\right), \tag{5.6}$$

meaning that whenever the posterior density $p(H'|\boldsymbol{z}_l;\boldsymbol{\theta})$ at the proposed sample is greater than the posterior density $p(H_k|\boldsymbol{z}_l;\boldsymbol{\theta})$ at the current sample, the proposed sample will automatically be accepted. If this is not the case it will be accepted with the probability $p(H'|\boldsymbol{z}_l;\boldsymbol{\theta})/p(H_k|\boldsymbol{z}_l;\boldsymbol{\theta})$. Considering Eq. 5.1, we can write the acceptance probability as

$$A(H'|H_t) = \min\left(1, \frac{\exp\big(-E(H', \boldsymbol{z}_l;\boldsymbol{\theta})\big)}{\exp\big(-E(H_k, \boldsymbol{z}_l;\boldsymbol{\theta})\big)}\right), \tag{5.7}$$

which can be feasibly evaluated.

Proposal Distribution

A common choice for the proposal distribution is a normal distribution centered at the current sample. In our case this is not possible because the rotational component of the pose lives on the manifold $SO(3)$, *i.e.* the group of rotations (see Section 3.1). We define $q(H'|H_t)$ implicitly by describing a sampling procedure and ensuring that it is symmetric. Ultimately this procedure will amount to a type of UARS-Normal distribution, as described in Section 3.2. [1]

We sample the translational and rotational component independently. The translational component $\boldsymbol{v}'$ of the proposed sample is directly drawn from a 3D isotropic normal distribution $\boldsymbol{v}' \sim \mathcal{N}(\boldsymbol{v}_k, \Sigma_{\boldsymbol{v}})$ centered at the translational component $\boldsymbol{v}_k$ of the current sample H_k. The rotational component is generated by drawing a random sample from a zero centered normal distribution $\boldsymbol{\omega}'_{R_k} \sim \mathcal{N}((0,0,0)^\intercal, \Sigma_R)$ in the tangent

[1] Note that we do not require the calculation of a probability density for the application as proposal distribution in the Metropolis algorithm. Therefore, the definition of the sampling procedure is sufficient.

space (see Section 3.1.4) at R_k. The new proposed rotation is then found by using Eq. 3.11 to map the vector onto the manifold

$$R' = \exp_{R_k}(\boldsymbol{\omega}'_{R_k}) = R_k \exp(\boldsymbol{\omega}'_{R_k}), \tag{5.8}$$

with $\exp(\cdot)$ defined in Section 3.1.2.

Initialization and Burn-in-phase

When the Metropolis algorithm is initialized in an area with low density it requires more iterations to provide a fair approximation of the expected value. To find a good initialization we run our inference procedure (described in the next section) using the current parameter set. We then perform the Metropolis algorithm for a total of 130 iterations, disregarding the samples from the first 30 iterations which are considered as burn-in-phase.

5.3.5 Inference Procedure

During test time, we aim at finding the MAP estimate, *i.e.* the pose maximizing our posterior density as given in Eq. (5.1). Since the denominator in Eq. (5.1) is constant for any given observation $\boldsymbol{z}$, finding the MAP estimate is equivalent to minimizing our energy function. To achieve this, we closely follow the original framework from [31] (see Chapter 2), but replace their scoring function with our energy, as described in Section 5.3.3.

5.4 Experiments

In the following, we compare our approach to the original framework (see Chapter 2) by Brachmann *et al.* for two different datasets. We first introduce the datasets. After that we describe details of our training procedure, and finally present quantitative and qualitative comparisons. We will see that we achieve considerable improvements for both datasets. Additionally, we observe that our CNN generalizes from a single training object to a set of 11 test objects, with large variability in appearance and geometry.

5.4.1 Datasets, Evaluation Protocol, Competitors

Datasets

We use two datasets featuring heavy occlusion. The first dataset was created by Brachmann *et al.* [31] by annotating the ground truth poses for eight partially occluded objects in images taken from the dataset of Hinterstoisser *et al.* [14]. We will refer to

this dataset as the *occlusion dataset* from [31] and [14]. It includes a total of 8992 test cases (1214 images with multiple objects), which are used for testing. We choose this dataset because it is more challenging than the original dataset from [14], on which [31] already achieves an average of 98.3% correctly estimated poses.

The second dataset was compiled by our selfs during the work on *System II* and published in [48] (see Section 4.4). We will refer to this dataset as *dataset of Krull* et al..

It provides six annotated RGB-D sequences of three different objects and consists of a total of 3187 images. We use three of the sequences for training and the other three (a total of 1715 test images) for testing.

Evaluation Protocol

We use the evaluation procedure as described in [31]. This means we calculate the percentage of correctly predicted poses for each sequence. As in [14], we calculate the average distance between the 3D model vertices under the estimated pose and under the ground truth pose. A pose is considered correct when the average distance is below 10% of the object diameter.

Competitors

We compare our method to the framework of [31] (see Chapter 2). Due to minor bugs in the original implementation from [31], there is a slight discrepancy between the results we obtained with a newer version and the originally published results. The numbers our new implementation achieves are slightly superior. In the following, we report two numbers, those of our fixed version and those of the method of [31], reported in [31]. As additional baseline, we provide the numbers from LineMOD [14] as reported in [31].

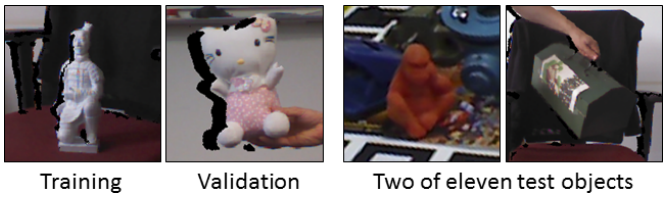


FIGURE 5.3: Images from one of our training-testing configurations: the *Samurai_1* sequence is used for training, the *Cat_1* for validation. Sequences of all objects are used for testing. Note, the objects are of vastly different shape and appearance.

5.4.2 Random Forests

For training and testing on the dataset of Krull *et al.* we use the random forest from [48] , previously used in *System II* (see Section 4.4). For the testing on the occlusion dataset we used a forest trained in [31], provided to us by the authors.

5.4.3 CNN Training Procedure

We trained three CNNs, each time using only a single object from the dataset of Krull *et al.*. The sequences *Toolbox_1*, *Cat_1*, and *Samurai_1* served as training sets (see Figure 5.3). The first 100 frames from *Samurai_1* were removed in order to obtain a high percentage of frames with occlusion. Our validation set consists of 100 randomly selected frames from the *Cat_1* sequence, or the *Samurai_1* sequence (in the case where *Cat_1* was used as training set). The weights of the CNN were randomly initialized. Before training, the random weights of the last layer were multiplied by factor 1000, in order to cover a greater range of possible energy values. After every 5th iteration of stochastic gradient descent, we perform inference on the validation set and adjust the learning rate. After training we pick the set of weights which achieved the highest percentage of correctly estimates poses on the validation set. We use the criterion from [14] to classify a pose as correct. One training cycle consisting of five steps of stochastic gradient descent and validation took[2] 9min 46sec (2min 27sec + 7min 19sec). Further details on our training procedure can be found in the Appendix (Section C.1).

5.4.4 Comparison

Occlusion Dataset from [31] and [14]

Quantitative results for this dataset are shown in Figure 5.4, for all individual test and training objects. Considering the average over all objects we achieve an improvement of up to 9.23% compared to our fixed version of [31] and **10.4%** compared to the reported values in [31]. Some qualitative results are illustrated in Figure 5.7. In Figure 5.5 we show another comparison of our method with respect to [31]. It illustrates that we achieve the biggest gain for occlusions between 50% and 60%.

Dataset of Krull *et al.*

For this dataset we observe similar results as with the previous dataset. Since the other sequences were used in training and validation, we evaluated only with the *Toolbox_2*, *Cat_2*, and *Samurai_2* sequences. When averaged over all objects we achieve

[2] We used an *Intel(R) Core (TM) i7-3820 CPU at 3.60GHz* with GeForce GTX 660 GPU. The *Cat_1* sequence was used for training and 100 random frames from *Samurai_1* for validation.

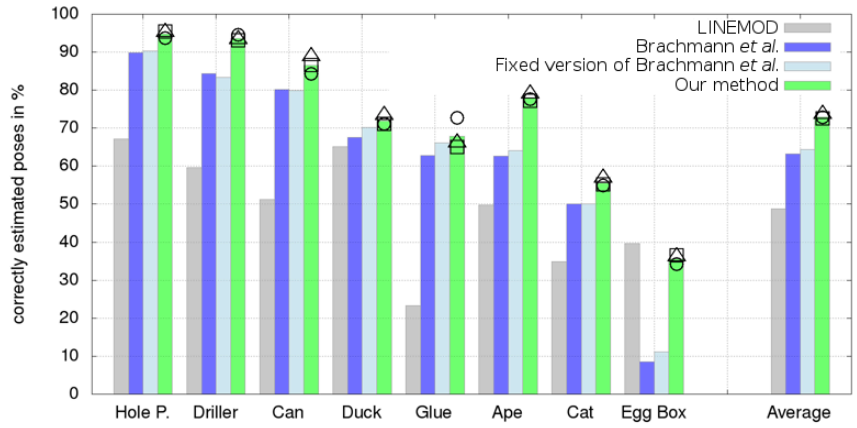


FIGURE 5.4: Quantitative comparison of our method against the results of [31] and LineMOD [14] on the *occlusion dataset* from [31] and [14]. *Circles*, *Squares*, and *Triangles* indicate the individual performance of CNNs trained with *Tool Box*, *Cat*, and *Samurai* respectively. The green bars indicate the average result. Averaged over all test and training objects we obtain the correct pose in **72.98%** of cases, in contrast to 63.24% for [31] and 48.84% for LineMOD [14]. A table with the detailed numbers can be found in the Appendix (Section C.2)

an improvement of **10.97%** compared to the results of [31]. The quantitative results can be found in Figure 5.6, and a few qualitative results are shown in Figure 5.8.

5.4.5 Discussion of Failure Cases

The failure cases which are framed red in Figure 5.7 have to be considered as failure of our learned energy function. However, the failure cases framed orange still exhibit a lower energy at the ground truth pose than at the estimate. This indicates a failure of the search procedure. It should be investigated in which cases the correct pose can be found using an alternative search procedure. In the dataset introduced by Krull *et al.* our accuracy for the *Tool Box* sequences is below the one of our competitor (see Figure 5.6). We attribute this to the fact that the *Tool Box* is the biggest object and most strongly affected by the down sampling schema described in Section 5.3.3.

5.5 Conclusion

We have presented a model for the posterior distribution in 6D pose estimation, which uses a CNN to map rendered and observed images to an energy value. We train the CNN-based on the maximum likelihood paradigm. It has been demonstrated that training on a single object is sufficient and the CNN is able to generalize to different objects and backgrounds. Our system has been evaluated on two datasets featuring heavy occlusion. By using our energy as objective function for pose estimation,

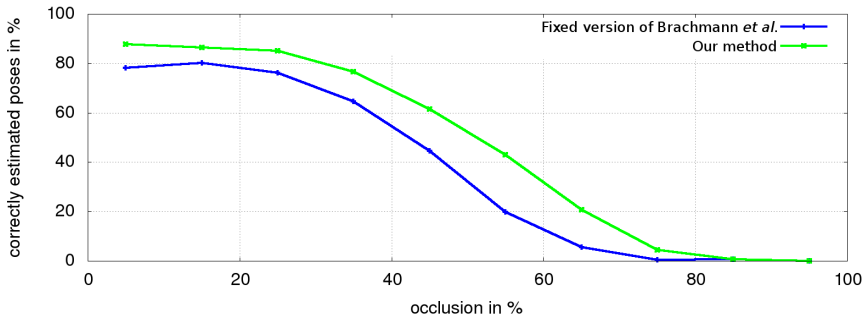


FIGURE 5.5: The percentage of correctly estimated poses for all test cases of the *occlusion dataset* from [31] and [14], as a function of the level of occlusion. For this we divided the test cases into bins according to the amount of occlusion, using a bin width of 10%. (See details of this procedure in the Appendix, in Section C.3.) We compare our method (using the CNN trained with the *Samurai* object) to our fixed version of [31]. We achieve improvements of over 20% for occlusion levels between 50% and 60%.

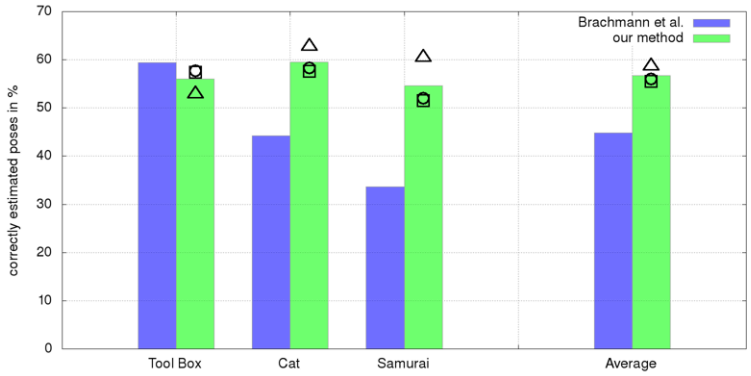


FIGURE 5.6: Comparison of our method on the dataset of Krull *et al.*, against the results of [31]. *Circles*, *Squares*, and *Triangles* indicate the individual performance of CNNs trained with *Tool Box*, *Cat*, and *Samurai* respectively. The green bars indicate the average result. We report 56.02%, 59.56%, and 54.65% correctly estimated poses for *Tool Box*, *Cat*, and *Samurai* respectively. Averaged over all test and training objects we achieve **56.74%**.

we were able to achieve considerable improvements compared to the best previously published results.

Our approach is not restricted to the feature channels and even the application we demonstrated. The architecture can in principle be applied to any kind of observed and rendered image. We think it would be worth investigating if the approach could be applied to other scenarios. An example could be pose estimation from pure RGB without recorded depth image and a forest to calculate features. Pose estimation for object classes could also benefit from our approach. Considering the recent success of CNNs in recognition [140, 141] it might be possible for a CNN to learn to compare observed images to renderings of an idealized model representing an object class instead of an instance. Our approach is not limited to comparing images of the same kind, as for example rendered and observed depth images. Instead, it could learn to assess the

plausibility of the shading in an observed RGB image by comparing it to a rendered depth image, which can be more easily produced than a realistic RGB rendering.

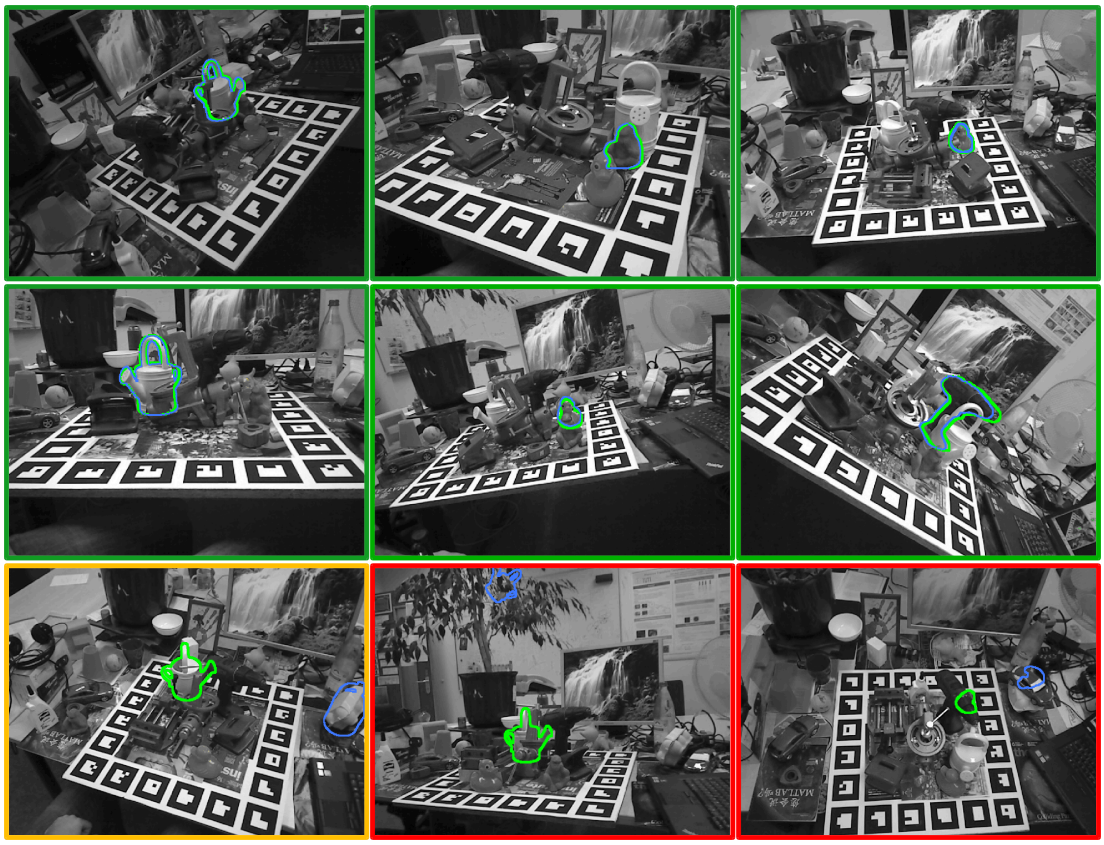


FIGURE 5.7: Qualitative results of our method on the *occlusion dataset* from [31] and [14]. Here, green and blue silhouettes correspond to the ground truth and our estimate, respectively. The test images depicted with a green frame show correct estimates. Images with orange and red frame show incorrect estimates. The image with an orange frame shows a case where the energy of the ground truth pose, according to Eq. (5.2), is lower than the energy of the estimated pose. In this case a better pose may be found with an improved optimization scheme.

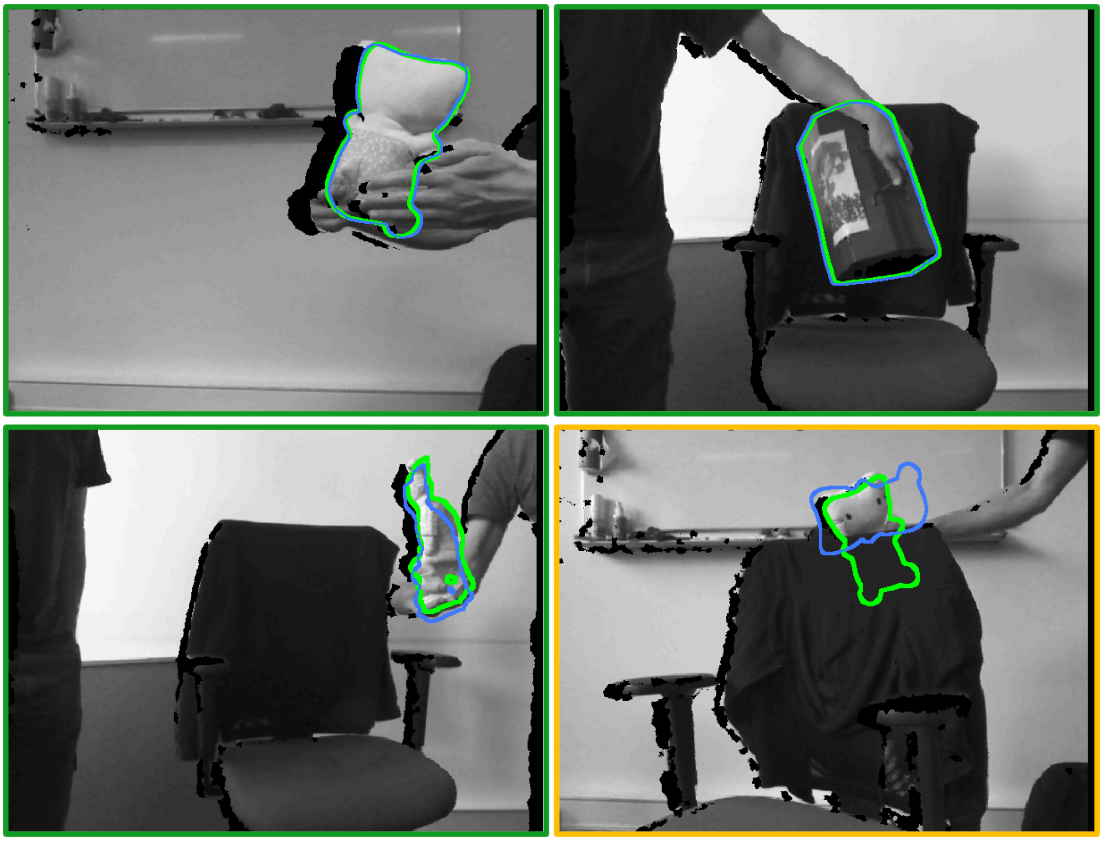


FIGURE 5.8: Qualitative results of our method on the test cases from the dataset introduced in [48]: Green frames correspond to correctly estimated poses according to the criteria from [14]. Orange frames correspond to incorrectly estimated poses with a lower energy at the ground truth than at the estimated pose.

Chapter 6

System III: Pose Estimation on a Budget

Contents

In this chapter we will discuss how to replace the static search procedure (see Section 2.3 and Figure 1.4 f) in the framework from [31] with a trained dynamic system that can learn to make optimal use of a limited computational budget. While the framework from [31] uses a fixed scheme to decide which hypotheses are to be refined and which one is ultimately chosen as final output, we will develop a system, termed *PoseAgent*, that iteratively decides which hypothesis should be refined next until a budget is exhausted. The behavior of the system is non-deterministic and governed by probability distributions. We train our system via sampling-based approach from RL, which allows for the use of gradient descent even though the pipeline includes non-differentiable steps. Note that the system was published in [82].

6.1 Introduction

Many tasks in computer vision can be seen as *learning a function*, usually learning to predict a desired output label given an input image. Advances in deep learning have led to huge progress in solving such tasks. In particular, CNNs work well when trained over large training sets using gradient descent methods to minimize the expected loss between the predictions and the ground truth labels.

However, important computer vision systems take the form of *algorithms* rather than being a simple differentiable function. For example sliding window search, superpixel partitioning, PFs, and classification cascades are algorithms realizing complex non-continuous functions.

The *algorithmic approach* is especially useful in situations where computational budget is limited: an algorithm can dynamically assign its budget to solving different aspects of the problem, for example, to take shortcuts in order to spend computation on more promising solutions at the expense of less promising ones.

We would like to *learn the algorithm*. Unfortunately, the hard decisions taken in most algorithmic approaches are non-differentiable, and this means that the structure and parameters of these efficient algorithms cannot be easily learned from data.

RL [33] offers a possible solution to learning algorithms. We view the algorithm as the *policy* of an RL agent, *i.e.* a description of dynamic sequential behavior. Then RL provides a framework to learn the parameters of such behavior with the goal of maximizing an expected reward, for example, the accuracy of the algorithm output.

We apply this perspective on the problem of 6D object pose estimation and use RL to learn the parameters of a deep algorithmic pipeline to provide the best possible accuracy given a limited computational budget.

Many of the applications for pose estimation such as robotics and augmented reality operate in a real-time setting and have to rely on a limited computational budget.

In this chapter, we will structure-wise build upon the previously discussed *System II*. While *System II* learns to assess the quality of pose hypotheses, it still relies on the original engineered search procedure from [31] (see Section 2.3). The procedure starts with a pool of hypotheses and then scores each of them. The subset of high-scoring hypotheses is refined and ultimately the highest-scoring hypothesis is returned as the pose estimate. Computationally the refinement step is quite expensive, and there is a trade-off between the number of refinements allowed and the expected quality of the result.

Ideally, one would train such state-of-the-art system end-to-end in order to learn how to use the optimal number of refinements to maximize the expected success of pose estimation. Unfortunately, treating the system as a black box with parameters to optimize is impossible for two reasons: (i) each selection process is non-differentiable

with respect to the scoring function; and (ii) the loss used to determine whether an estimated pose is correct is also non-differentiable.

We recast pose estimation as an RL problem in order to overcome these difficulties. We model the search procedure as an RL agent which we call *PoseAgent*. PoseAgent is granted more flexibility than the original system: it is given a fixed budget of refinement steps, and is allowed to manipulate its hypothesis pool by selecting individual poses for refinement, until the budget is spent.

In our PoseAgent model each agent decision follows a probability distribution over possible actions. This distribution is called the *policy* and we can differentiate and optimize this continuous policy through the *stochastic policy gradient* approach [154]. As a result of this stochastic approach the final pose estimate becomes a random variable, and each run of PoseAgent will produce a slightly different result.

This stochastic policy gradient approach is very general and does not require differentiability of the used loss function. As a consequence we can directly take the gradient with respect to the expected loss of interest, *i.e.* the number of correctly estimated poses.

Training in stochastic policy gradient can be difficult due to the sampling induced additional variance of the estimated gradients [154, 155].

To overcome this problem we propose an efficient training algorithm that radically reduces the variance during training compared to a naïve technique.

We compare our approach to *System II* and achieve improvements in accuracy, while using the same or a smaller average number of refinement steps.

6.1.1 Contributions

In summary, the main contributions presented in this chapter are:

- To the best of our knowledge, we are the first to apply a policy gradient approach to the object pose estimation problem.
- Our approach allows the use of a non-differentiable reward function corresponding to the original evaluation criterion.
- We present an efficient training algorithm that dramatically reduces the variance during training.

6.2 Related Work

Below, we first discuss the relation between the previously discussed *System II* and *System III* (the PoseAgent), which we are about to present. Then, we will provide a short review of RL methods used in a setting similar to ours.

6.2.1 Relation to System II

The system we will present in this chapter is closely related to *System II*. As *System II*, we follow the general framework from [31] and only focus on a single component of the pipeline. We now however, go a step further. *System II* has replaced the scoring function (see Section 2.4 and Figure 1.4 g) with a learned probabilistically motivated energy (see Section 5.3.3). An examination of the failure cases (Section 5.4.5) suggests the results might be further improved by optimizing the search procedure (Section 2.3 and Figure 1.4 f). We will now describe a way to learn this search procedure.

The system we will present in this chapter is also methodologically similar to *System II*. Here, we use a similar CNN construction as in *System II*, feeding both rendered and observed image patches into to a CNN. Also, in *System II*, as in the system we will present now, the CNN output is interpreted as part of a probability distribution.

These distributions however fulfill a different function, and are trained differently. The training process in *System II* is seen as learning the posterior distribution, which is then maximized during testing using the fixed search procedure. In contrast, here we will present a training process that directly modifies the behavior of the agent, in order to maximize the number of correctly estimated poses.

6.2.2 Reinforcement Learning in Similar Tasks

RL has traditionally been successful in areas like robotics [156], control [157], advertising [158], network routing [159], or playing games [160]. While the application of RL seems natural for these tasks, RL is increasingly being successfully applied in computer vision systems where the interpretation of the system as an agent interacting with an environment is not always so intuitive. In [161] RL has been used to visually parse pictures of building facades. Recent papers apply RL for 2D object detection and recognition [162–165]. However, we are to our knowledge the first to apply RL for 6D object pose estimation.

In [162, 163], an agent shifts its area of attention over the image until it makes a final decision. Instead of moving a single 2D area of attention over search space like [162, 163], we work with a pool of multiple 6D pose hypotheses. The agent in [164] focuses its attention by moving a 2D fixation point, though operates on a set of precomputed image regions to gather information and make a final decision. Our agent instead manipulates its hypothesis pool by refining individual hypotheses.

Caicedo *et al.* [163] use Q-learning, in which the CNN predicts the quality of the available state-action pairs. Mnih *et al.* [162] and Mathe *et al.* [164] use a different RL approach based on stochastic policy gradient, in which the behaviour of the agent is directly learned to maximize an expected reward. We follow [162, 163] in using stochastic policy gradient.

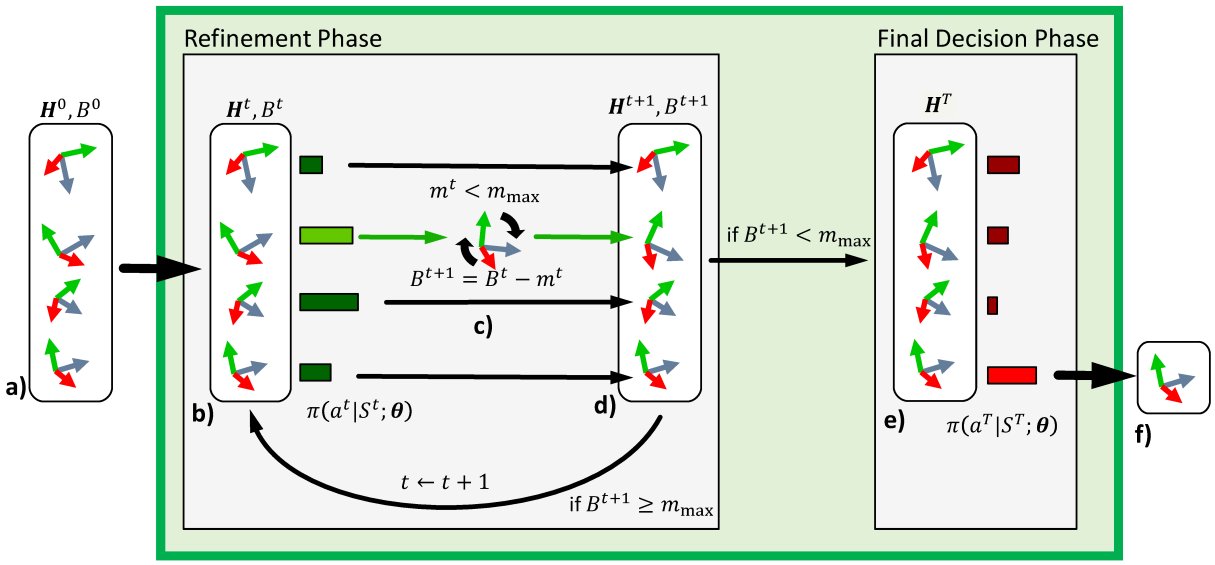


FIGURE 6.1: The PoseAgent search procedure: **a):** The initial pool of hypotheses is sampled and handed to the agent. **b):** The agent selects a hypotheses $H^t_{a^t}$ by sampling from the policy $\pi\left(a^t|S^t;\boldsymbol{\theta}\right)$. **c):** The selected hypothesis is refined until convergence, or until the maximum number of iterations is exceeded. The budget is reduced accordingly. **d):** If enough budget is left, the refinement phase continues. If the budget is exhausted a final decision phase begins. **e):** The final selection is made by sampling from the policy $\pi\left(a^T|S^T;\boldsymbol{\theta}\right)$. **f):** The selected pose $H^T_{a^T}$ is output as final pose estimate.

6.3 Method

In this chapter, we consider the one-shot pose estimation problem, as defined in Section 1.1.1. We will start by describing PoseAgent, our RL agent. Then, we will discuss how to train our agent, introducing our new, efficient training algorithm.

6.3.1 PoseAgent

We will now describe our RL agent, PoseAgent, and how it performs its search. An overview of the process can be found in Figure 6.1. The agent operates in two phases: (i) the refinement phase, in which the agent chooses individual hypotheses to undergo refinement; and (ii) the final decision phase, in which it has to decide which pose should be selected as final output. In the following, we will discuss both phases in detail.

The search begins with the **refinement phase**. The pose agent starts with a pool $\boldsymbol{H}^0 = (H^0_1 \dots H^0_N)$ of hypotheses which have been generated as described in Section 2.2, and a fixed budget B^0 of possible refinement steps.

At each time step t, the agent chooses one hypothesis index a^t, which we call an action. [1] The chosen hypothesis is refined and the next time step begins. We limit the maximum number of times the agent may choose the same hypothesis for refinement to $\tau_{\max}$. Hence, over time, the pool of actions (resp. hypotheses) the agent may choose for refinement decreases. We denote the set of possible actions $A^t = \{a \in \{1, \dots, N\} | \tau_a^t < \tau_{\max}\}$, where τ_a^t denotes how many times hypothesis a has been chosen for refinement before time t. Subsequently, the agent modifies the hypothesis pool by refining hypothesis $H_{a^t}^{t+1} = g(H_{a^t}^t)$, where $g(\cdot)$ is the refinement function. All other hypotheses remain unchanged $H_a^{t+1} = H_a^t \; \forall a \neq a^t$.

We perform refinement as follows (see also Section 2.3.1): We render the object in pose $H_{a^t}^t$. Each pixel within the rendered mask is tested for being an inlier. All inlier pixels are used to re-calculate the pose with the Kabsch algorithm. We repeat this procedure multiple times for the single, chosen hypothesis until the number of inlier pixels stops increasing or until the number m^t of executed refinement steps exceeds a maximum m_{max}. The budget is decreased by the number of refinement steps performed, $B^{t+1} = B^t - m^t$. The agent proceeds choosing refinement actions until $B^t < m_{\text{max}}$, in which case further refinement may exceed the total budget B^0 of refinement steps.

When this point has been reached, the refinement phase terminates, and the agent enters the **final decision phase**, in which the agent chooses a hypothesis as the final output. We denote the final action as $a^T \in \{1 \dots N\}$ and the final pose estimate as $\hat{H} = H_{a^T}^T$. The agent receives a reward of $r = 1$ in case the pose is correct or a negative reward of $r = -1$ otherwise. As in Chapters 4 and 5, we use the pose correctness criterion from [14].

In the following, we describe how the agent makes its decisions. During both, the refinement phase and the final decision phase, the agent chooses from the hypothesis pool. We describe the agent behavior by a probability distribution $\pi\left(a^t | S^t; \boldsymbol{\theta}\right)$ referred to as policy. Given the current state S^t, which contains information about the hypothesis pool and the observation $\boldsymbol{z}$, the agent selects a hypothesis by drawing a sample from the policy. The vector $\boldsymbol{\theta}$ of learnable parameters consists of CNN weights (described in Section 6.3.1). We will first give details on the state space S^t before describing the policy $\pi\left(a^t | S^t; \boldsymbol{\theta}\right)$.

State Space

We model our state space in a way that allows us to use our new, efficient training algorithm, described in Section 6.3.2. We assume that the current state S^t of the hypothesis pool decomposes as $S^t = (s_1^t, \dots s_N^t)$, where s_a^t will be called the state of

[1] Note that we use the upper index $\cdot^t$ to denote the algorithmic time steps in the search procedure as opposed to the lower index notation $\cdot_t$, used to indicate time in the tracking formulation of Section 1.1.2.

hypothesis H_a^t, or simply the hypothesis state. The state of an hypothesis contains the original input observation $\boldsymbol{z}$, the pose hypothesis H_a^t, as well as a vector $\boldsymbol{f}_a^t$ of additional context features of the hypothesis (see Section 6.3.1). In summary, this gives $s_a^t = (\boldsymbol{z}, H_a^t, \boldsymbol{f}_a^t)$.

Policy

Our agent makes its decisions using a softmax policy. The probability of choosing a particular action a^t during the refinement phase is given by

$$\pi\left(a^t | S^t; \boldsymbol{\theta}\right) = \frac{\exp\left(-E(s_{a^t}^t; \boldsymbol{\theta})\right)}{\sum_{a \in A^t} \exp\left(-E(s_a^t; \boldsymbol{\theta})\right)}, \tag{6.1}$$

where $E(s_a^t; \boldsymbol{\theta})$ will be called the energy of the state s_a^t. We will abbreviate it as $E_a^t = E(s_a^t; \boldsymbol{\theta})$. The energy of a state in the softmax policy is a measure of how desirable it is for the agent to refine the hypothesis. We use the same policy in the final decision phase, but with a different energy $E'(s_a^t; \boldsymbol{\theta})$ abbreviated by ${E'}_a^t$. We use a CNN to predict both energies, E_a^t and ${E'}_a^t$. In the next section, we discuss the structure of the CNN and how it governs the behavior of the agent.

CNN Architecture

We give an overview of the CNN architecture used in this work in Figure 6.2. As in *System II*, the CNN compares rendered and observed images. We use the same six input channels as in *System II* (see Section 5.3.3), namely: the rendered depth channel, the observed depth channel, a rendered segmentation channel, the object probability channel, a depth mask, and a single channel holding the difference between object coordinates.

There are however two major differences in our CNN compared to the one used in *System II*. Firstly, while *System II* predicts a single energy value of a pose, we jointly predict two separate energy values: one energy E_a^t for the refinement phase and one energy $E_a'^t$ for the final decision phase.

Secondly, we input additional features to the network by concatenating them to the first fully connected layer. The features are: The number of times the hypothesis has already been selected for refinement, The distance the hypothesis has moved during its last refinement and the average distance of the hypothesis before refinement to all other hypotheses in the original pool.

Our CNN consists of the following layers: 128 kernels of size $6 \times 3 \times 3$, 256 kernels of size $128 \times 3 \times 3$, a 2×2 *max-pooling* layer, 512 kernels of size $256 \times 3 \times 3$, a max-pooling operation over the remaining size of the image, finally 3 fully connected layers. The

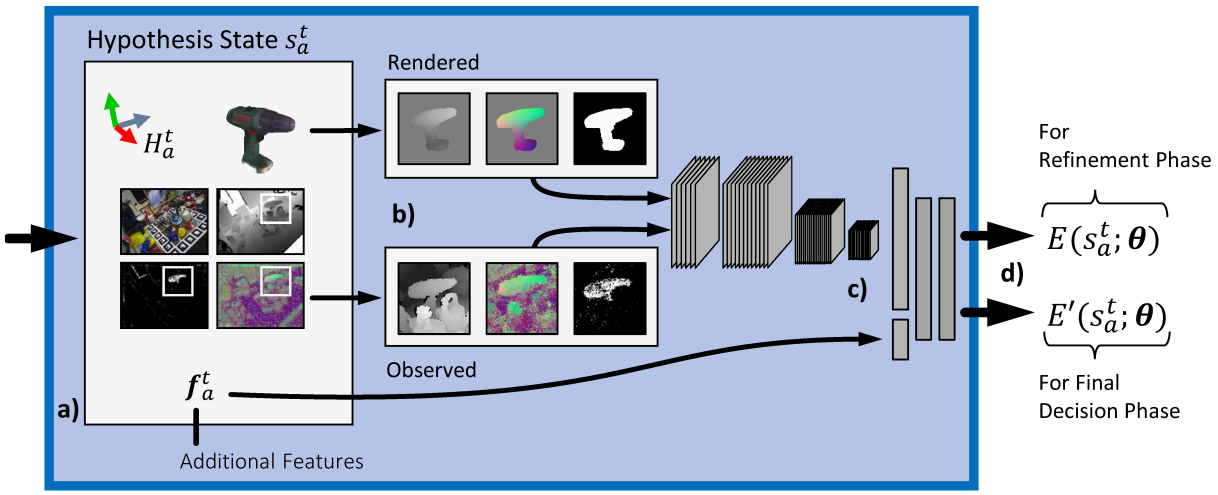


FIGURE 6.2: CNN energy calculation: **a):** The system takes an hypothesis state s_a^t as input. It consists of a pose hypothesis H_a^t, the observed images $\boldsymbol{z}$, and the additional features $\boldsymbol{f}_a^t$ encoding the context and history of the pose. **b):** We use the hypothesis to render the object and to cut out patches in the observed images. **c):** The images are processed by the CNN. After multiple convolutional layers the result is concatenated with the features $\boldsymbol{f}_a^t$ and further processed by fully connected layers. **d):** The network outputs two energy values: $E(s_a^t)$ to be used in the refinement phase and $E'(s_a^t)$ to be used in the final decision.

features $\boldsymbol{f}_a^t$ are concatenated to the first fully connected layer, as shown in Figure 6.2. Each layer, except the last is followed by a *tanh* operation.

6.3.2 Policy Gradient Training

We will now discuss the training procedure for our PoseAgent. First, we will give a general introduction of policy gradient training, and then apply the approach to the PoseAgent. Finally, we will introduce an efficient algorithm that greatly reduces variance during training and makes training feasible.

The goal of the training is the maximization of the expected reward $\mathbb{E}[r]$. This expected value depends on the environment as well as on the policy of our agent. In stochastic policy gradient methods one attempts to approximate the gradient with respect to the policy parameters $\boldsymbol{\theta}$. Note that since we are dealing with the expected value it becomes possible calculate derivatives, even if the reward function itself is non-differentiable. By making use of the general equality $\frac{\partial}{\partial\theta_j}p(x;\theta_j) = p(x;\theta_j)\frac{\partial}{\partial\theta_j}\ln p(x;\theta_j)$, we can write the derivative of the expected reward with respect to each parameter θ_j in $\boldsymbol{\theta}$ as

$$\frac{\partial}{\partial\theta_j}\mathbb{E}[r] = \mathbb{E}\left[r\frac{\partial}{\partial\theta_j}\ln p(s^{1:T}, a^{1:T};\boldsymbol{\theta})\right], \tag{6.2}$$

where $p(s^{1:T}, a^{1:T}; \boldsymbol{\theta})$ is the probability of a particular sequence of states $s^{1:T} = (s^1 \dots s^T)$ and actions $a^{1:T} = (a^1 \dots a^T)$ to occur.

Because of the Markov property of the environment, it is possible to decompose the probability and rewrite it as

$$\frac{\partial}{\partial \theta_j} \mathbb{E}\left[r\right] = \mathbb{E}\left[r \sum_{t=0}^{T} \frac{\partial}{\partial \theta_j} \ln \pi\left(a^t | S^t; \boldsymbol{\theta}\right)\right]. \tag{6.3}$$

Following the REINFORCE algorithm [80], we approximate Eq. 6.3 using sampled sequences $(S_k^{1:T_k}, a_k^{1:T_k})$, generated by running the agent, as described in Section 6.3.1, on training images,

$$\frac{\partial}{\partial \theta_j} \mathbb{E}\left[r\right] \approx \frac{1}{K} \sum_{k=1}^{K} r_k \sum_{t=0}^{T_k} \frac{\partial}{\partial \theta_j} \ln \pi\left(a_k^t | S_k^t; \boldsymbol{\theta}\right), \tag{6.4}$$

where T_k is the number of steps in the sequence and r_k is the reward achieved in the sequence.

Efficient Gradient Calculation

We will now introduce an algorithm (Algorithm 1), to dramatically reduce the variance of estimated gradients, by allowing us to use a higher number of sequences K, in a given time. The basic idea is to make use of the special decomposable structure of the state space and our policy. The advantage of our algorithm compared to the naïve implementation is illustrated in Figure 6.4.

Starting from a hypothesis pool $\boldsymbol{H}^0$, only a finite number $N \cdot (\tau_{\max} + 1)$ of different hypothesis states $s_a^\tau | a \in \{1, \dots N\}, \tau \in \{0, \dots \tau_{\max}\}$ can occur during a run of our PoseAgent. Here, $s_a^\tau = (H_a^\tau, \boldsymbol{f}_a^\tau)$ shall denote the state of hypothesis a after it has been refined τ times.

The algorithm pre-computes all possibly occurring hypothesis states s_a^τ, and predicts all corresponding energy values [2] E_a^τ in advance using the CNN.

While this comes with some computational expense, it allows us to rapidly sample large numbers of sequences without having to re-evaluate the energy function.

To illustrate why this is possible, let us now reconsider the calculation of the derivatives in Eq. 6.4. Using the chain rule, we can write them as

$$\frac{\partial}{\partial \theta_j} \ln \pi\left(a^t | S^t; \boldsymbol{\theta}\right) = \sum_{a \in A^t} \frac{\partial E_a^t}{\partial \theta_j} \frac{\partial}{\partial E_a^t} \ln \pi\left(a^t | S^t; \boldsymbol{\theta}\right), \tag{6.5}$$

[2] To improve readability, we will not differentiate between E_a^τ and $E_a'^\tau$ in this section.

where

$$\frac{\partial}{\partial E_a^t} \ln \pi \left(a^t | S^t; \boldsymbol{\theta}\right) = \begin{cases} \pi \left(a^t | S^t; \boldsymbol{\theta}\right) - 1 & \text{if } a = a^t \\ \pi \left(a^t | S^t; \boldsymbol{\theta}\right) & \text{else} \end{cases} . \tag{6.6}$$

We can now rearrange Eq. 6.4 as sum over possible hypothesis states

$$\sum_{\tau=0}^{\tau_{\max}} \sum_{a=1}^{N} \frac{\partial E_a^\tau}{\partial \theta_j} \frac{1}{K} \underbrace{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbb{1}\left(\tau_{a,k}^t = \tau\right) \frac{\partial}{\partial E_a^\tau} \ln \pi \left(a_k^t | S_k^t; \boldsymbol{\theta}\right) r_k}_{D(a,\tau)} . \tag{6.7}$$

Here, $\mathbb{1}\left(\tau_{a,k}^t = \tau\right)$ is the indicator function. It has the value 1 only when the hypothesis a at time t in sequence k has been selected for refinement exactly $\tau_{a,k}^t = \tau$ times. It has the value 0 in any other case.

Our algorithm works by first calculating the inner sums in Eq. 6.7 and storing the results in the entries $D(a, \tau)$ of a table D. We compute these sums with a single iteration over all sequences k and all time steps t. The accumulation of these values is computationally cheap, because it does not require any rendering or involvement of the CNN.

This structure allows us to increase the number of sampled sequences K without much computational cost. The algorithm can process an arbitrary amount of sequences using only a single back propagation pass of the CNN for each possible hypothesis state s_a^t. In a naïve implementation, the number of required forward-backward passes would increase linearly with the number of sampled sequences.

Let us look at the algorithm in detail. It consists of three parts:

- **Initialization Phase:** We generate the original hypothesis pool as described in Section 2.2. Then, we refine all hypotheses $\tau_{\max}$ times and predict the energy values E_a^τ for all of them using the CNN.

- **Sampling Phase:** We sample sequences $(s_k^{1:T}, a_k^{1:T})$ as described in Section 6.3.1, using the precomputed energies. We observe the reward r_k for each sequence. Then, we calculate for each time t, each selected hypothesis a_k^t and each possible hypothesis a the derivative $\frac{\partial}{\partial E_a^\tau} \ln \pi \left(a_k^t | S_k^t; \boldsymbol{\theta}\right) r_k$ using Eq. 6.6. We accumulate the results in the corresponding table entries $D(a, \tau_{a,k}^t)$. This corresponds to the inner sums in Eq. 6.7.

- **Gradient Update Phase:** We once more process each of the hypothesis states s_a^τ with the CNN and use standard back propagation to calculate $\frac{\partial E_a^\tau}{\partial \theta_j}$. We multiply the results with $D(a, \tau)$ and accumulate them up in another table G to obtain the final gradients. This corresponds to the outer sums in Eq. 6.7.

Algorithm 1: Efficient Gradient Calculation

Initialization Phase:

generate hypothesis pool $\boldsymbol{H}^0$;
refine each hypothesis $\tau_{\max}$ times;
calculate and store E_a^τ;
initialize table entries $D(a,\tau) \leftarrow 0$ and $G(j) \leftarrow 0$

Sampling Phase:

for $k = 1 : K$ **do**
  sample path $(s_k^{1:T_k}, a_k^{1:T_k})$ using E_a^τ ;
  receive reward r_k;
  for $t = 1 : T_k, a = 1 : N$ **do**
    $D(a,\tau_{a,k}) \leftarrow D(a,\tau_{a,k}) + \frac{\partial}{\partial E_a^\tau} \ln \pi\left(a_k^t | S_k^t; \boldsymbol{\theta}\right) r_k$
  end
end

Gradient Update Phase:

for $\tau = 0 : \tau_{\max}\ ,\ a = 1 : N$ **do**
  calculate $\frac{\partial E_a^\tau}{\partial \theta_j}$ via back propagation;
  for all CNN parameters j **do**
    $G(j) \leftarrow G(j) + \frac{\partial E_a^\tau}{\partial \theta_j} \frac{1}{K} D(a,\tau)$;
  end
end

Output: $G(j) \approx \frac{\partial}{\partial \theta_j} \mathbb{E}\left[r\right]$;

6.4 Experiments

In the following we will describe the experiments to compare our method to the baseline of *System II*. Our experiments confirm that our learned search procedure is able to use its budget in a more efficient way, compared to *System II*. It outperforms the baseline system, while using on average a smaller number refinement steps.

Additionally we will describe an experiment regarding the efficiency of our training algorithm compared to a naïve implementation of the REINFORCE algorithm. We find that our algorithm can dramatically reduce the gradient variance during training.

We conducted our experiments on the dataset of Krull *et al.* (see Section 4.4) introduced in [48]. It features six RGB-D sequences of hand held sometimes strongly occluded objects.

We use the same division as in *System II* (see Section 5.4.3), reserving the sequences *samurai_1*, *cat_1* and *toolbox_1* for training and validation, while testing on the remaining *samurai_2*, *cat_2* and *toolbox_2* sequences.

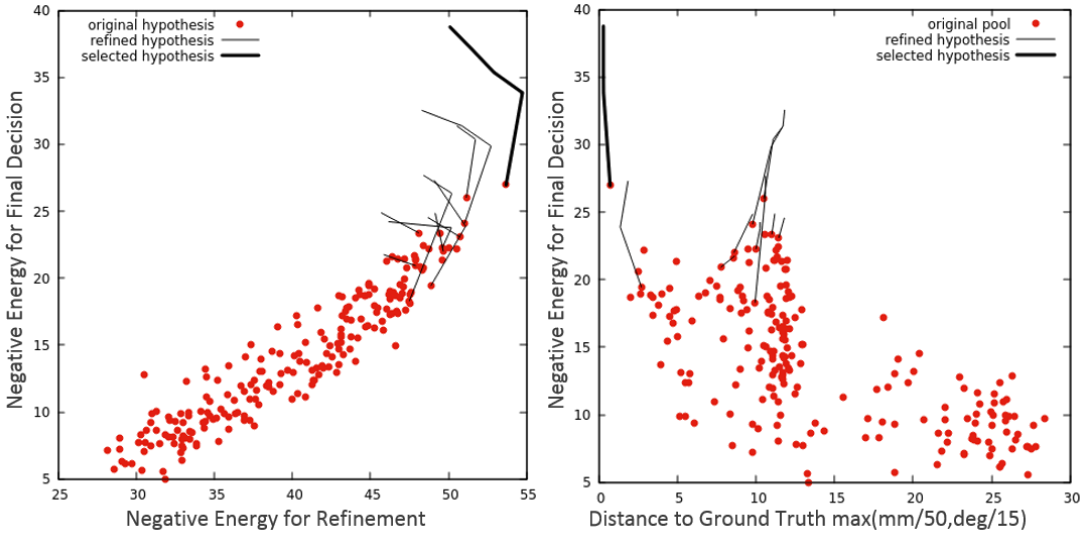


FIGURE 6.3: Visualization of PoseAgent behavior while processing a single image. Every red dot corresponds to one hypothesis from the original pool. Every line corresponds to the path of a hypothesis that has been refined by the agent. The bold line corresponds to the path of the hypothesis that as been selected during the final decision phase. A high negative energy for refinement $-E(s_a^t; \boldsymbol{\theta})$ leads to a high probability of being selected during the refinement phase. A high negative energy for the final decision $-E'(s_a^t; \boldsymbol{\theta})$ leads to a high probability of being selected during the final decision phase. **Left:** For most hypotheses the negative energy for refinement first increases and at some point decreases again, while the negative energy for final decision continues to increase. This allows the agent to investigate different hypotheses, instead of focusing on a single promising candidate. **Right:** The agent successfully selects the hypotheses close to the ground truth. The hypothesis closest to the ground has the highest negative energy for the final decision.

6.4.1 Training and Validation Procedure

We train our system on the *samurai_1* sequence. As *System II* (see Section 5.4.3) we omit the first 400 frames in the training sequence to achieve a higher percentage of occluded images.

We train our system with two different parameter settings: Using a hypothesis pool size of $N = 210$, which is the setting used in *System II*, and a larger pool size of $N = 420$.

To determine the adequate size of the budget B of refinement steps, we ran *System II* on our validation set and determined the average number of refinement steps it used. We set our budget during training to the resulting number $B = 77$.

During training we allow a hypothesis to be chosen $\tau_{\max} = 3$ times for refinement. We set the maximum number of refinement steps per iteration to $m_{\max} = 10$.

Using stochastic gradient descent, we go through our training images in random order and run Algorithm 1 to approximate the gradient. We sample $K = 50k$ sequences for every image l, and use an additional $50k$ to estimates a baseline b. We

perform a parameter update after every image. Starting with an initial learning rate of $\lambda^0 = 25 \cdot 10^{-4}$, we reduce it according to $\lambda^l = \lambda^0/(1 + l\nu)$, see [166], with $\nu = 0.01$. We use a fixed momentum of 0.9.

We skip all images in which none of the hypotheses from the pool leads to a correct pose after being refined $\tau_{\max}$ times. The reason for this is that such an image will always produce a reward of $r = -1$ and cannot provide meaningful information.

We also skip all images in which more than 10% of the hypotheses from the pool lead to a correct pose. Such images contribute little, because they are easily solved. We augment our training data, by randomly deciding for every image whether to mirror the CNN input along the x- and y-axis.

We run the training procedure for 96 hours on an *Intel E5-2450 2.10GHz with Nvidia Tesla K20x GPU* and save a snapshot every 50 training images. To avoid over-fitting, we test these saved snapshots on our validation set and choose the model with the highest accuracy.

In order to reduce the computational time during validation, we considered only images in which the object was at least 5% occluded[3].

6.4.2 Additional Baselines

Two demonstrate the advantage of dynamically distributing a given computational budget, we implemented two cut-down versions of PoseAgent, which serve as additional baselines. The baseline method abbreviated as *RandRef* randomly selects a hypothesis to refine at every iteration. When the budget is exhausted, it chooses the hypothesis with the best predicted final selection energy $E'(s_a^t; \boldsymbol{\theta})$.

The baseline *BestRef* directly picks the hypothesis with the best $E(s_a^t; \boldsymbol{\theta})$, refines it until the budget is exhausted and outputs it as final decision. We used the best performing settings when running the baselines: (τ_{max} = 6, m_{max} = 5) for pool size N = 210 and (τ_{max} = 7, m_{max} = 4) for pool size N = 420.

6.4.3 Testing Conditions

We compared both versions of our model, using $N = 210$ and $N = 420$, against *System II*, using the corresponding pool size. In all experiments with the baseline method *System II*, we use the identical CNN with the original weights trained as described in Section 5.4.3.

Apart from the pool size, we used the identical testing conditions as in *System II* (see Section 5.4), including the same random forest originally trained in [31]. To classify a pose in correct or false we use the same point-distance-based criterion used in

[3] see Appendix Section C.3

System II. A pose is considered correct, when the average distance between the vertices of the 3D model in the ground truth pose and the evaluated pose is below a threshold.

While the number of refinement steps in our setting is restricted, *System II* does not provide any guarantees on how many refinement steps are used. To ensure a fair comparison, we first ran *System II* and recorded the average number of refinement steps that it required on each test sequence. When running our method, we set the budget for each sequence to this recorded value, making sure that PoseAgent could never use more refinement steps than *System II*. The total average number of refinement steps required by both methods can be found Table 6.1 and 6.2.

We evaluate our method using different parameters for $\tau_{\max}$ and $m_{\max}$, so that $\tau_{\max} \cdot m_{\max} \approx 30$. Meaning that a each pose can have an approximate maximum of 30 refinement steps. A higher value of $\tau_{\max}$ (and lower value of $m_{\max}$) means that PoseAgent can make more fine grained decisions on where to spend its budget. We use the following combinations for the two values: ($\tau_{\max}$=3,$m_{\max} = 10$), ($\tau_{\max} = 5, m_{\max} = 6$), ($\tau_{\max} = 6, m_{\max} = 5$), ($\tau_{\max} = 7, m_{\max} = 4$).

6.4.4 Results

		Ours					
	System II	$\tau_{\max}$=3	$\tau_{\max}$=5	$\tau_{\max}$=6	$\tau_{\max}$=7	RandRef	BestRef
Cat 2	63.05	67.81	68.61	**71.52**	68.74	45.17	49.27
Samurai 2	**60.30**	51.66	53.32	54.82	51.66	31.73	34.88
Toolbox 2	52.96	52.07	**60.06**	54.44	59.76	35.50	32.84
Total	60.06	58.94	61.47	**62.18**	60.88	38.47	40.88
Avg. ref. steps	68.71	62.94	65.91	66.60	67.20		

TABLE 6.1: Percent correct poses using a hypothesis pool of $N = 210$

		Ours					
	System II	$\tau_{\max}$=3	$\tau_{\max}$=5	$\tau_{\max}$=6	$\tau_{\max}$=7	RandRef	BestRef
Cat 2	72.98	74.70	74.97	76.29	**78.01**	54.17	56.03
Samurai 2	**66.45**	59.30	59.63	58.80	61.13	39.87	40.86
Toolbox 2	64.79	65.38	68.93	**71.60**	71.01	52.07	53.85
Total	68.03	67.37	68.32	69.14	**70.62**	48.67	50.21
Avg. ref. steps	71.12	65.00	68.04	68.66	69.39		

TABLE 6.2: Percent correct poses using a hypothesis pool of $N = 420$

The results of our experiments can be seen in Table 6.1 and 6.2. PoseAgent is able to improve the best published results on the dataset by a total of **10.56%**. When we compare our method only to the baseline working on the same hypothesis pool size

we are still able to outperform it. With the original pool size of $N = 210$ by **2.12%** and the increased pool size of $N = 420$ by **2.59%**.

Note that the budget is set in a way that is extremely restrictive, ensuring that PoseAgent can never use more refinement steps than *System II* uses on average.

In both settings ($N = 210$ and $N = 420$) there appears to be a trend that an increase of $\tau_{\max}$, which corresponds to a more fine grained control of PoseAgent, leads to an improvement in accuracy. The only exception here is $\tau_{\max} = 7$ in the $N = 210$ setting. It should be noted that PoseAgent was trained with a different setting of $\tau_{\max} = 3$ and was able to generalize to the different settings used during testing.

We measured the average run time of the method (using CPU rendering) to be between 17 (Samurai 2) and 34 (Cat 2) seconds per image on an Intel E5-2450 2.10GHz with NVidia Tesla K20x GPU using a hypothesis pool of $N = 420$.

6.4.5 Efficiency of the Training Algorithm

In order to investigate the efficiency of out training algorithm compared to a naïve implementation of the REINFORCE algorithm, we conducted the following experiment:

We ran our training algorithm as well as the naïve implementation up to 100 times on a single training image without updating the network.

To estimate the variance of the gradient, we calculated the standard deviation of 1000 randomly selected elements from the resulting gradient vector of the CNN and averaged them. We recorded the required computation time to process the image on an Intel E5-2450 2.10GHz with Nvidia Tesla K20x GPU.

The process was repeated for $K = 5$, $K = 50$, $K = 500$, $K = 5000$ and $K = 50000$ sequences in case of the efficient algorithm. In case of the naïve implementation we used $K = 1$, $K = 2$, $K = 3$ and $K = 4$ sequences. To keep the computation time in reasonable limits we used a reduced setting with a hypothesis pool size of $N = 21$ for both methods. As can be seen in Figure 6.4 our algorithm allows us to reduce variance greatly with almost no increase in computation time.

6.5 Conclusion

We have demonstrated a method to learn the algorithmic search procedure in a pose estimation system using a policy gradient method. Our system learns to make efficient use of a given budget and is able to outperform the original system, while using on average less computational resources. We have presented an efficient algorithm for the gradient approximation during training. The algorithm is able to sharply reduce gradient variance, without a significant increase in computation time.

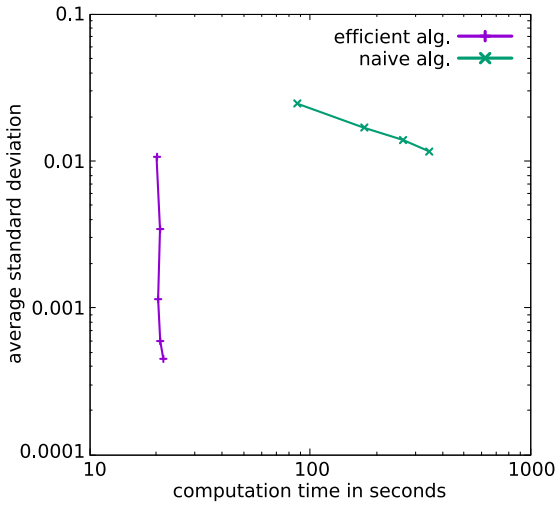


FIGURE 6.4: Observed gradient variance during training as a function of time: Our method is able process dramatically more sequences with almost no increase in computation time compared to a naive implementation of the REINFORCE algorithm. The result is a drastically reduced gradient variance

We see multiple interesting future directions of research based on PoseAgent. (i) As sensible extension of the method could use a budget of computation time, instead of refinement steps. (ii) One could investigate a soft version of PoseAgent, which is not working with a fixed budget, but can instead decide what is the appropriate time to stop. In such systems the used computational budget can be part of the reward function. Such formulations have been used in the context of CNNs [167–169]. (iii) The sequential structure of the current system does not allow simple parallelization, but a PoseAgent that learns to do search, while making use of multiple computational cores could be conceived.

Chapter 7

Discussion and Future Work

Contents

In this final chapter we will take another look at different aspects of the presented systems. We will compare them under different perspectives, highlighting their methodological differences, and suggesting potential combinations as well as possible future research topics.

7.1 Pose Domain versus Hypothesis Domain

The systems presented in this thesis operate in different domains. *Systems I* uses hypotheses as part of its proposal distribution, but ultimately operates directly on the 6D Pose space. *Systems II* models its posterior distribution in pose space, but performs its estimation on the discrete hypothesis set. The agent in *Systems III* is trained and operates only on the discrete set of pose hypotheses.

We see two advantages of using an hypotheses-based approach. First, by generating hypotheses from predicted correspondences, we make use of geometric knowledge. We agree with the argument from [46] that such knowledge should be used, when possible instead of attempting to learn it from data. Second, hypotheses provide the ability for a direct comparison between alternatives. We assume it to be easier to choose between a set of alternatives than to search directly in the 6D continuous pose space. The underlying assumption here is of course that the refinement procedure can find the correct pose, starting from the original hypothesis set. If this is not the case, maybe due to extreme occlusion, *Systems II* and *III* must fail, while *Systems I* still has a chance to succeed.

A logical step to ensure than an hypothesis, close enough to the correct solution is included, would be to learn the process of hypothesis generation as well. We think that RL could again be an appropriate tool to achieve this.

The current hypothesis generation procedure (see Section 2.2) selects one pixel and then two more in its vicinity to generate a hypothesis. The pixels are sampled using the predicted object probabilities as guidance. One possibility would be to learn these object probabilities in an end-to-end fashion. As a matter of fact, in [46] Brachmann *et al.* present an RL-based system that can learn object coordinates in a similar way. It would be a natural extension to learn the object probabilities as well.

Alternatively, one could go a step further and learn the sampling process itself. There are trade-offs to be considered in the way the three pixels are selected. For example, if the second and third pixels are in close proximity to the first one, they are more likely to be part of the same object. However, since the prediction is noisy the predicted pose will be more accurate if the pixels are further apart. It would be desirable to try and understand this trade-off in an analytical way in terms describing the expected error. We can also see how an RL agent could be trained to choose the next pixel-based on the previously selected one or two.

We think it is also worth considering whether the hypotheses in the original pool should really be generated independently of one another, as it is currently the case. It seems that the chance of finding the correct pose could be increased if the hypotheses were distributed more diversely, avoiding the creation of densely packed clusters. There exists quite a body of work on the topic of generating diverse solutions and hypotheses, e.g. [170–172]. An RL agent generating pose hypothesis could build on such findings. It could prune the pool of hypotheses to avoid concentration by using *determinantal point processes* [170] or a similar model, or it could generate the pool incrementally, considering the distance to previously found hypotheses in each step.

7.2 Maximum Likelihood versus Reward Maximization

Even though *Systems II* and *III* are structurally similar–both use a CNN to compare rendered and observed images–they rely on radically different training paradigms.

System II is trained using the maximum likelihood approach. It is known that maximizing likelihood is equivalent to minimizing the *Kullback–Leibler divergence* [173]. We ultimately adjust the parameters $\boldsymbol{\theta}$ to align the distribution described by our CNN with the distribution provided by our training data. This kind of approach to learning is attractive because of its flexibility. When we are learning by matching probability distributions we are not committed to a particular loss function during testing. Once our probability distribution is trained we can use it for whatever purpose we choose. We can find an estimate by maximizing an expected loss during test time, or we can

use the distribution as part of a larger probabilistic model and reason with it. One of the core features of probability theory is, after all, the possibility of combining distributions in a principled way. A HMM, as the one used in $System\ I$, can for example be constructed with trained distributions for observation likelihood and motion.

The approach taken in *System III* on the other hand is not based on matching probability distributions. Instead, the RL agent is trained to simply optimize the expected reward. This is very much akin to the empirical loss-based training of CNNs that was so successful in recent years [40, 41, 174, 175]. In *System III* we have to specify the reward function during training, and we ultimately train a specialized algorithm to perform exactly the task we specified through the reward. In this respect, the approach is much less flexible, compared to the training of probability distributions. On the other hand, loss-based RL allows for the end-to-end training of very complex algorithmic procedures that can even include discrete decisions.

We think it is worth considering how to combine these two paradigms in a beneficial way. A very common approach is to use a probabilistic model, but to train it not by matching distributions, but by minimizing an expected loss calculated on the final estimate. In methods such as [176–178] CRF models are trained through a differentiable inference procedure. Their CNN-based potentials are tuned to minimize an expected loss after inference. Discriminatively trained tracking systems in the HMM framework are another example: In methods such as [179–181] the authors train probabilistic models for tracking, by minimizing an expected loss of the tracking result.

We see another approach for combining the two philosophies. Instead of training a probabilistic model with its inference procedure in a loss-based fashion, it is also possible to train an RL agent as part of a probabilistic model. Recently, there has been some work going in this direction. Systems such as [167–169] use CNNs to model probability distributions over classes and consider RL to make efficient use of computational resources by dynamically deciding which part of a CNN should be activated or deactivated, or how information should flow within the network. We see this as an interesting line of research and will discuss a possible application in Section 7.4.

7.3 Particle Filer and Learned Posteriors

Here we would like to point out the possibility of combining a trained posterior distribution, as the one from *System II*, with the PF framework from *System I* or similar probabilistic tracking systems.

In [181] Burkhart *et al.* call attention to the fact that the filtering equation Eq. 4.3 can be reformulated as

$$p(\mathcal{X}_{t+1}|\boldsymbol{z}_{1:t+1}) \propto p(\boldsymbol{z}_{t+1}|\mathcal{X}_{t+1}) \int p(\mathcal{X}_{t+1}|\mathcal{X}_t) p(\mathcal{X}_t|\boldsymbol{z}_{1:t}) d\mathcal{X}_t \tag{7.1}$$

$$\propto \frac{p(\mathcal{X}_{t+1}|\boldsymbol{z}_{t+1})}{p(\mathcal{X}_{t+1})} \int p(\mathcal{X}_{t+1}|\mathcal{X}_t) p(\mathcal{X}_t|\boldsymbol{z}_{1:t}) d\mathcal{X}_t. \tag{7.2}$$

The proportionality here is meant in the sense that the two terms are equal up to a factor, depending only on the observation $\boldsymbol{z}_{t+1}$, but not on the state $\mathcal{X}_{t+1}$. This has implications for HMM-based tracking systems in general, as well as for our PF-based *System I*. By additionally assuming a uniform prior $p(\mathcal{X}_{t+1})$ over the pose, we can substitute the observation likelihood used in the computation of the weights in our particle filter with the trained unnormalized posterior $p(\mathcal{X}_{t+1}|\boldsymbol{z}_{t+1}) = \exp\left(-E(H_{t+1}, \boldsymbol{z}; \boldsymbol{\theta})\right)$ (Eq. 5.1) distribution from *System II*. Note that a normalization is not required, as the normalization constant would be identical for all particles.

Due to the computational effort required for the application of the CNN, it might be problematic to apply this idea directly in a real-time system. However, we still think that the general idea is worth considering for two reasons: First, it might be possible to reduce the required amount of computation (see Section 7.4). Second, the approach could also be applied for an offline tracking formulation, in which computation time is not as essential.

As pointed out in [181], the training or construction of observation models is often difficult. Many PF methods (e.g. [37, 38, 101, 102, 127], but also *System I*) use heuristically constructed observation models that are not motivated or trained in a meaningful way. We would like to argue that training a model for the posterior and applying it in a tracking framework is a more principled approach than the use of a heuristically build observation model.

We think that the general scheme of training an energy-based model for the posterior, as described in *System II*, to later use it in a HMM-based framework, is applicable in settings beyond object pose estimation.

7.4 Particle Filer and Reinforcement Learning

In our opinion the combination of the PF tracking framework with the RL approach bears an interesting potential. For online tracking systems the efficient use of computational resources can be essential, as a missed frame due to slow computation can lead to further errors at later points in time. RL could be a tool to help distribute computational resources in a sensible way. In a particle filter often the evaluation of the observation likelihood, especially in a formulation as described in Section 7.3, for all

particles is the computational bottleneck. Consequently, we believe that a PF system could benefit by using RL-based techniques such as [167–169] to dynamically decide how much computational effort should be spent on the computation of the observation likelihood of each individual particle. It might be possible to quickly assign a low observation likelihood to some of the particles, while others might require more attention.

The potential for improvement could be even greater, if we model not the computation of observation likelihoods as an RL agent, but the PF as a whole. Instead of individually deciding how much effort to spend on each particle, such an agent could learn to iteratively distribute computational budget among particles. One could go a step further and allow the dynamic adjustment of the number of particles or of the proposal distribution itself. Such an approach could be seen as an extension of the work on discriminative PF tracking [179–181], with the additional possibility of learning discrete choices in the procedure.

7.5 Conclusion

In this thesis, we have taken a probabilistic stance on the problem of object pose estimation, a task with enormous potential impact on the modern world. We have demonstrated that a probabilistic perspective can help us to address the difficult challenges of visual ambiguity and restricted computational resources, arising in this task. Building on the state-of-the-art framework of [31], we developed and evaluated three systems, each applying a different flavor of probabilistic modeling and sampling-based implementation to a different aspect of the original framework.

In *System I* we have shown how to convert the one-shot framework to a real-time PF-based pose tracking system, build on a probabilistic model. The system includes a novel proposal distribution to make sensible use of the different elements in the original framework. In *System II* we have presented a CNN to evaluate the quality of pose hypotheses by learning the comparison of rendered and observed images. We have trained the CNN as part of a probability distribution. This approach potentially allows for the use in a larger probabilistic model such as the tracking framework of *System I*. In *System III* we constructed an RL agent that learns to make optimal use of a restricted computational budget during the search for the correct pose. The agent's non-deterministic behavior allows for the training of an algorithmic search procedure involving discrete choices.

The methods we presented have improved the state-of-the-art in object pose estimation. Furthermore, we believe that they have the potential to be applied in problems beyond pose estimation as well.

Appendices

Appendix A

Derivations Regarding SO(3)

A.1 Derivation of the Density Factor Between Tangent Spaces in $SO(3)$

We will now derive the factor $\phi(\psi_{R_0,R})$ from Eq. 3.19 that is to be applied when comparing probability densities defined in two different tangent spaces of $SO(3)$. The factor is depicted as a function of $\psi_{R_0,R}$ in Figure A.1. A differently motivated derivation of the factor can be found in [107].

A.1.1 Definition of the Problem

Let us assume we have a probability density $p_{R_0}(\boldsymbol{\omega}_{R_0})$ defined in the tangent space at the location R_0. Because of the symmetric structure of SO(3) we can, without loss of generality, assume that R_0 is the identity rotation. By using Eq. 3.13, we can map elements $\boldsymbol{\omega}_{R_0}$ from the tangent space at R_0 to the tangent space at a different location R. The resulting elements in this space will be denoted by $\boldsymbol{\omega}_R = f_{R_0 \to R}(\boldsymbol{\omega}_{R_0})$, with $f_{R_0 \to R}$ defined in Eq. 3.13. By applying this mapping to elements distributed according to $p_{R_0}(\boldsymbol{\omega}_{R_0})$, we can thus indirectly define a distribution with probability density $p_{R_0 \to R}(\boldsymbol{\omega}_R)$ over the elements of the tangent space at R.

Let us assume now that we are interested in the probability density at the particular location $\boldsymbol{\omega}_{R_0} = \log_{R_0}(R)$ (see Eq. 3.12), *i.e.* at the location corresponding to R. By definition, the corresponding representation of $\boldsymbol{\omega}_{R_0}$ in the tangent space at R will lie at the origin: $\boldsymbol{\omega}_R = f_{R_0 \to R}(\boldsymbol{\omega}_{R_0}) = (0,0,0)^\intercal$.

We are now interested in calculating the ratio of probability densities at this location

$$\phi(\psi_{R_0,R}) = \frac{p_{R_0 \to R}(\boldsymbol{\omega}_R)}{p_{R_0}(\boldsymbol{\omega}_{R_0})} = \frac{p_{R_0 \to R}\left((0,0,0)^\intercal\right)}{p_{R_0}(\boldsymbol{\omega}_{R_0})}. \tag{A.1}$$

This factor will allow us to derive the density in the tangent space at R from the density in the tangent space at R_0 as

$$p_{R_0 \to R}\,(0,0,0)^\intercal = p_{R_0}(\boldsymbol{\omega}_{R_0})\phi(\psi_{R_0,R}). \tag{A.2}$$

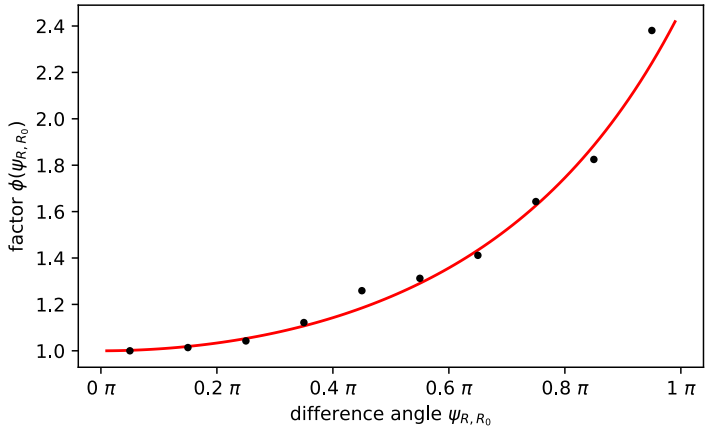


FIGURE A.1: The factor $\phi(\psi_{R_0,R})$ for the comparison of densities defined in different tangent spaces. The **red line** is calculated according to Eq. 3.19. **Black circles** are the result of a Monte Carlo approximation: We sampled 5000 points in the tangent space at R_0 and transformed (via Eq. 3.13) them to a tangent space at a location R. We then calculated the ratio of the number of points that fall into spherical volumes (of diameter 10^{-5}) located at $\log_{R_0}(R)$ in the tangent space at R_0 and at the origin in the tangent space at R. We used importance sampling with an isotropic normal proposal distribution centered at $\log_{R_0}(R)$ with a standard deviation of $2 \cdot 10^{-5}$

Here, $\psi_{R_0,R}$ is the angle of the difference rotation RR_0^{-1} between R and R_0 (see Eq. 3.5). If we assume, as stated above, that R_0 is the identity rotation then $\psi_{R_0,R}$ is simply the angle of the rotation R. To simplify the notation in the following derivation, we will denote it as ψ, with $\psi = \psi_{R_0,R}$

A.1.2 Derivation

Note that, because of the symmetry of $SO(3)$, the factor $\phi(\psi)$ depends only on the angle ψ and not on the axis of the rotation nor its direction. Thus, we can assume without loss of generality for R to be a rotation around the first axis with angle $0 < \psi < \pi$:

$$R_\psi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{pmatrix}. \tag{A.3}$$

We will from now on view it as a function of ψ and denote it as R_ψ to make this explicit. By using the *change of variables technique* [182] the factor $\phi(\psi)$ can be derived as

$$\left|\det\left(J_{f_{R_\psi \to R_0}}(\boldsymbol{\omega}_{R_\psi})\right)\right|, \tag{A.4}$$

where $J_{f_{R_\psi \to R_0}}(\boldsymbol{\omega}_{R_\psi})$ is the Jacobian of the map $f_{R_\psi \to R_0}(\boldsymbol{\omega}_{R_\psi})$ from the tangent space at R_ψ to the tangent space at R_0, *i.e.* the inverse of $f_{R_0 \to R_\psi}(\boldsymbol{\omega}_{R_0})$. Based on Eq. 3.14 and

on the assumption we made earlier that R_0 is the identity rotation, we can calculate

$$\boldsymbol{\omega}_{R_0} = f_{R_\psi \to R_0}(\boldsymbol{\omega}_{R_\psi}) = \log\left(\exp(\boldsymbol{\omega}_{R_\psi}) R_\psi\right), \tag{A.5}$$

and proceed to calculate the Jacobian as

$$J_{f_{R_\psi \to R_0}}(\boldsymbol{\omega}_{R_\psi}) = \begin{pmatrix} \frac{\partial \omega^1_{R_0}}{\partial \omega^1_{R_\psi}} & \frac{\partial \omega^2_{R_0}}{\partial \omega^1_{R_\psi}} & \frac{\partial \omega^3_{R_0}}{\partial \omega^1_{R_\psi}} \\ \frac{\partial \omega^1_{R_0}}{\partial \omega^2_{R_\psi}} & \frac{\partial \omega^2_{R_0}}{\partial \omega^2_{R_\psi}} & \frac{\partial \omega^3_{R_0}}{\partial \omega^2_{R_\psi}} \\ \frac{\partial \omega^1_{R_0}}{\partial \omega^3_{R_\psi}} & \frac{\partial \omega^2_{R_0}}{\partial \omega^3_{R_\psi}} & \frac{\partial \omega^3_{R_0}}{\partial \omega^3_{R_\psi}} \end{pmatrix}, \tag{A.6}$$

where $\omega^1_{R_\psi}, \omega^2_{R_\psi}, \omega^3_{R_\psi}$ and $\omega^1_{R_0}, \omega^2_{R_0}, \omega^3_{R_0}$ shall denote the components of $\boldsymbol{\omega}_{R_\psi}$ and $\boldsymbol{\omega}_{R_0}$ respectively.

As stated in Section A.1.1, we need to determine the partial derivatives in Eq. A.6 at the origin of the tangent space $\boldsymbol{\omega}_{R_\psi} = (0,0,0)^\intercal$. However, we find that they are not defined at this location. Instead, we determine the limits of Eq. A.6, with $\omega^1_{R_\psi}, \omega^2_{R_\psi}$, and $\omega^3_{R_\psi}$ approaching zero. We find the following limits:

$$\hat{J}_{f_{R_\psi \to R_0}}(\boldsymbol{\omega}_{R_\psi}) = \lim_{\boldsymbol{\omega}_{R_\psi} \to (0,0,0)^\intercal} J_{f_{R_\psi \to R_0}}(\boldsymbol{\omega}_{R_\psi}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{\psi(\cos(\psi)+1)}{2\sin(\psi)} & \frac{\psi}{2} \\ 0 & -\frac{\psi}{2} & \frac{\psi(\cos(\psi)+1)}{2\sin(\psi)} \end{pmatrix}. \tag{A.7}$$

Finally, we calculate the determinant

$$\phi(\psi) = \left|\det\left(\hat{J}_{f_{R_\psi \to R_0}}(\boldsymbol{\omega}_{R_\psi}\right)\right| = \left(\frac{\psi/2}{\sin(\psi/2)}\right)^2 = \frac{\psi^2}{2(1-\cos\psi)}. \tag{A.8}$$

A.2 Derivation of UARS Density in Tangent Space

To derive the UARS density in tangent space given in Eq. 3.22, we start with the corresponding distribution in spherical coordinates. A uniform distribution of rotation axis can be described by the joint distribution over azimuth $-\pi < \varphi < \pi$ and elevation $0 < \vartheta < \pi$ angles.

$$p(\vartheta, \varphi) = \frac{1}{4\pi}\sin(\vartheta). \tag{A.9}$$

Since the angle ψ is sampled independently, we can write the joint distribution of all three variables as product. Making use of the fact that $C(\psi;\kappa)$ is symmetric about 0,

we can restrict $0 < \psi < \pi$ by including the factor 2:

$$p(\vartheta, \varphi, \psi; \kappa) = \frac{1}{2\pi} \sin(\vartheta) C(\psi|; \kappa). \tag{A.10}$$

We now have to convert the density from spherical to Cartesian coordinates $\boldsymbol{\omega}_{R_0} = (\omega^1_{R_0}, \omega^2_{R_0}, \omega^3_{R_0})^\intercal$. This can again be achieved by multiplying a factor $\tilde{\phi}(\psi)$, found via the change of variables technique [182], similarly as in Section A.1.2. We can map spherical coordinates to Cartesian coordinates as

$$\begin{aligned} \omega^1_{R_0} &= \psi \sin(\vartheta) \cos(\varphi) && \text{(A.11)} \\ \omega^2_{R_0} &= \psi \sin(\vartheta) \cos(\varphi) && \text{(A.12)} \\ \omega^3_{R_0} &= \psi \cos(\vartheta). && \text{(A.13)} \end{aligned}$$

To find the conversion factor, we require the inverse map, which can be calculated as

$$\begin{aligned} \psi &= |\boldsymbol{\omega}_{R_0}| && \text{(A.14)} \\ \vartheta &= \arccos(\omega^3_{R_0}/|\boldsymbol{\omega}_{R_0}|) && \text{(A.15)} \\ \varphi &= \arctan((\omega^2_{R_0})/\omega^1_{R_0}). && \text{(A.16)} \end{aligned}$$

Note that here we assume $\omega^1_{R_0} > 0$. We can do this without loss of generality because of the symmetric nature of the problem. Since the final density is rotationally symmetric, and will be identical for all $\boldsymbol{\omega}_{R_0}$ with $|\boldsymbol{\omega}_{R_0}| = \psi$ we can do all calculations at the location

$$\omega^1_{R_0} = \psi, \;\; \omega^2_{R_0} = 0, \;\; \omega^3_{R_0} = 0. \tag{A.17}$$

We now calculate the Jacobi matrix containing the derivatives of ψ, ϑ, φ in the directions of $(\omega^1_{R_0}, \omega^2_{R_0}, \omega^3_{R_0})$. We do our calculation at the location given by Eq. A.17. We find:

$$J_{\text{cart}\to\text{sphere}}(\boldsymbol{\omega}_{R_0}) = \begin{pmatrix} 1 & 0 & \\ 0 & 0 & -\frac{1}{\psi} \\ 0 & \frac{1}{\psi} & 0 \end{pmatrix} \tag{A.18}$$

The resulting factor is

$$\tilde{\phi}(\psi) = |\det\left(J_{\text{cart}\to\text{sphere}}(\boldsymbol{\omega}_{R_0})\right)| = \frac{1}{\psi^2} \tag{A.19}$$

By multiplying Eq. A.10 with the factor, we obtain the following tangent space density in Cartesian coordinates:

$$sin(\vartheta)\frac{C(\psi = |\boldsymbol{\omega}_{R_0}|; \kappa)}{2\pi(\psi = |\boldsymbol{\omega}_{R_0}|)^2}. \tag{A.20}$$

By using Eq. A.15 we obtain $\vartheta = \frac{\pi}{2}$ for the location chosen in Eq. A.17. As mentioned above the density is identical for all $\boldsymbol{\omega}_{R_0}$ with $|\boldsymbol{\omega}_{R_0}| = \psi$. We can thus calculate it as

$$p_{R_0}^{\text{UARS}}(\boldsymbol{\omega}_{R_0}; R_0, \kappa) \quad = \quad \frac{C(\psi = |\boldsymbol{\omega}_{R_0}|; \kappa)}{2\pi(\psi = |\boldsymbol{\omega}_{R_0}|)^2}. \tag{A.21}$$

Appendix B

Further Details on System I

B.1 Details on Fitting the Continuous Distribution

During the construction of our proposal distribution we fit a continuous distribution $f_{\mathrm{UARS-N}}(H; \tilde{H}^{\mathrm{center}}, \tilde{\Sigma}, \tilde{\kappa})$ to a set of extrapolated particles $\tilde{P}_{t+1}$. We will now describe the fitting process in detail.

To find $\tilde{H}^{\mathrm{center}} = (\tilde{R}^{\mathrm{center}}, \tilde{\boldsymbol{v}}^{\mathrm{center}})$ We calculate the translational component $\tilde{\boldsymbol{v}}^{\mathrm{center}}$ as the mean translation of $\tilde{P}_{t+1}$, and the rotational component $\tilde{R}^{\mathrm{center}}$ as the mean rotation, (see Section 3.1.3).

We then use $\tilde{\boldsymbol{v}}^{\mathrm{center}}$ to calculate the covariance matrix $\tilde{\Sigma}$ of the translational components of $\tilde{P}_{t+1}$. To find our estimate $\tilde{\kappa}$ for the concentration parameter $\tilde{\kappa}$ we calculate the difference angles $\psi_{\tilde{R}^i_{t+1}, \tilde{R}^{\mathrm{center}}}$ (see Eq. 3.5) between $\tilde{R}^{\mathrm{center}}$ and each rotation $\tilde{R}^i_{t+1}$ in $\tilde{P}_{t+1}$. Since we do not know the direction of the rotations we apply a random sign to each $\psi_{\tilde{R}_i, \tilde{R}^{\mathrm{center}}}$ and finally compute $\tilde{\kappa}$ via Eq. 4 from [183]. Note that we add an additional constant $c = 10^{-4}$ to the denominator for numerical stability.

B.2 Hypothesis Generation

In the construction of our proposal distribution *System I* uses a sampling scheme similar to the one in framework from [31] (see Section 2.2).

We will now describe in detail where our procedure differs. The sampling process used in the framework draws the first random pixel i_1 from the entire image according to $p_{c,i}$, followed by two more in its vicinity. We in contrast, consider only pixels inside a square window around the projected center of $H^{\mathrm{center}} = (R^{\mathrm{center}}, \boldsymbol{v}^{\mathrm{center}})$. We calculate the width of the window as $f\delta_c / -\boldsymbol{v}_z^{\mathrm{center}}$, where $\boldsymbol{v}_z^{\mathrm{center}}$ is the z-coordinate of $\boldsymbol{v}^{\mathrm{center}}$.

In the framework from [31] an error is calculated for each sampled hypothesis by mapping the predicted object coordinates of the three pixels into camera space using the generated hypothesis and comparing them with the observed 3D camera coordinates. Hypotheses are accepted, if all errors are below 5% of the object diameter.

We use a different error measure, based on the assumption that the distance between two points in camera and object space should be identical. We calculate the Euclidean distance between each possible pair of points out of the three. We do this in camera and object space and for each pair compute the difference between the two. Our error measure is defined as the maximum of these differences. We accept a hypothesis whenever its error is smaller than the objects diameter.

While in the framework from [31] hypothesis generation is repeated until 210 hypotheses are accepted, we always sample 500 times. In cases where less than 5 hypotheses are accepted, we stop calculation of the global estimate and use only the local estimate.

B.3 List of Parameters

The following parameters were used in all experiments:

Training Parameters	
maximum feature offset:	20 pixel meters
number of features generated at each node:	1000
ratio of 'da-d' to 'da-rgb' features	0.5
number of trees:	3
random pixels per image to learn tree structure:	1000
random pixels per image to learn leaf distributions:	5000
stopping criterion: minimum number of pixels per node:	50

General Testing Parameters	
number of particles K:	70
control parameter λ_{like}:	20
depth comp. weight λ_{depth}:	10
coordinate comp. weight λ_{coord}:	2
object comp. weight λ_{obj}:	10
threshold τ_d used in $E\text{depth}(H)$:	50 mm
threshold τ_d^{occ} used in $E\text{depth}(H)$:	30 mm
threshold τ_y used in $E\text{coord}(H)$:	$(0.2\ \delta_c)^2$
number of Hypothesis to be sampled:	500
threshold used during sampling of poses:	0.05 δ_c
inlier threshold used in refinement:	20 mm

Proposal Distribution Parameters	
covariance matrix for prop. dist. Σ^{prop}:	$I\sigma_{\boldsymbol{v}}^{\text{prop}}$
std of random translation in prop.dist. $\sigma_{\boldsymbol{v}}^{\text{prop}}$:	2 mm
std of random rotation in prop.dist. $\sigma_R^{\text{prop}} = 1/\sqrt{\kappa^{\text{prop}}}$:	0.01rad
maximum iterations for refinement of H_{t+1}^{global}:	15
maximum iterations for refinement of H_{t+1}^{global}:	15
maximum iterations for final optimization H_{t+1}^{global}:	30
mixture weight in prop. α^{prop}:	0.5

The following motion model parameters were used in all experiments performed on the dataset of Choi and Christensen:

Motion Model Parameters (Dataset from Choi and Christensen)	
damping parameter for translation $\lambda_{\boldsymbol{v}}$:	0.7
damping parameter for translation λ_R:	0.6
std of random translation $\sigma_{\boldsymbol{v}}^{\mathrm{mm}}$:	14 mm
std of random rotation in motion model $\sigma_R^{\mathrm{mm}} = 1/\sqrt{\kappa^{\mathrm{mm}}}$:	0.05rad

The following motion model parameters were used in all experiments performed on the dataset of Krull *et al.*:

Motion Model Parameters (Dataset dataset of Krull *et al.*)	
damping parameter for translation $\lambda_{\boldsymbol{v}}$:	0.7
damping parameter for translation λ_R:	0.6
std of random translation $\sigma_{\boldsymbol{v}}^{\mathrm{mm}}$:	20 mm
std of random rotation in motion model $\sigma_R^{\mathrm{mm}} = 1/\sqrt{\kappa^{\mathrm{mm}}}$:	0.15rad

B.4 Approximate UARS Density

Throughout the experiments described in Chapter 4, we used only an approximate density to describe the UARS distribution. We effectively use

$$\tilde{f}_{\mathrm{UARS}}(R; R_0, \kappa) = C \left(\arccos \left(\frac{1}{2} (\mathrm{tr}(R_0^{-1} R) - 1) \right); \kappa \right). \tag{B.1}$$

Compared to the correct density given in Eq. 3.27 we were thus lacking the factor

$$\frac{4\pi}{3 - \mathrm{tr}(R_0^{-1} R)}. \tag{B.2}$$

Only later considerations have led us to the Eqs. 3.26 and 3.27 The benefit of using these equations remains to be evaluated in future experiments.

Appendix C

Further Details on System II

C.1 Further Details on our Training Procedure

We will now discuss the training procedure of our CNN. We use the *Torch 7* framework to implement our network. Our training procedure starts with a randomly initialized set of network parameters. We use the random initialization provided by *Torch 7*. To cover a greater range of possible energy values from the beginning, we multiply the weights in the last layer by 1000 before training starts.

During training we repeat the following steps:

1. Randomly pick five training samples from the training set.
2. Perform a gradient step for each of the training samples:
   (a) Calculate partial derivatives $\frac{\partial}{\partial \theta_j} E\left(H_l, \boldsymbol{z}_l; \boldsymbol{\theta}^l\right)$ of the energy at the ground truth pose H_l using back propagation.
   (b) Run the inference scheme as described in Section 5.3.5 on the training image $\boldsymbol{z}_l$.
   (c) Use the result as initialization for Metropolis sampling described in Section 5.3.4.
   (d) Calculate the partial derivatives $\frac{\partial}{\partial \theta_j} E\left(H_k, \boldsymbol{z}_l; \boldsymbol{\theta}^l\right)$ of the energy at each pose sample H_k and average the results.
   (e) Calculate the gradient of the log likelihood according to Eq. 5.4.
   (f) Calculate the new parameter set $\boldsymbol{\theta}^{l+1}$ using the gradient and current learning rate.
3. Use the current set of parameters $\boldsymbol{\theta}^l$ to perform inference on the validation set and determine the number of correctly estimated poses.
4. Update the learning rate similar to [166] as:
   $\lambda^l = 10^{-5}\lambda^0/(1 + \lambda^0 \nu l)$

In the end we pick the set of parameters which performed best on the validation set according to the number of correctly estimated poses. In the case where two parameter sets perform equally well, we pick the later one.

In our experiments we iterated the scheme described above 48 times and used the parameters $\lambda^0 = 10$ and $\nu = 0.1$. For the proposal distribution in the Metropolis sampling scheme we used the covariance matrix $\Sigma_{\boldsymbol{v}} = 25^{-1} I_3$ to sample of the translational component and the covariance matrix $\Sigma_R = 0.01^{-1} I_3$ to sample the rotational components.

In order to increase the number of training samples we randomly decide in each training step whether to rotate all images by 180deg.

C.2 Detailed Experimental Results

Dataset by Hinterstoisser [14] and Brachmann [31]

Here we provide a table with detailed results on the dataset by Hinterstoisser [14] and Brachmann[31]. In Table C.1, we show the percentage of correctly estimated poses for the different objects and the average percentage. The results correspond to Figure 5.4. The best results for each object are printed in bold numbers.

	Brachmann *et al.* [31]	Fixed Version of Brachmann *et al.*	*System II*
Ape	62.6%	64.1%	**79.06%**
Can	80.2%	79.95%	**88.9%**
Cat	50%	50.05%	**56.84%**
Driller	84.3%	83.43%	**93.32%**
Duck	67.6%	70.12%	**73.4%**
Egg Box	8.5%	11.17%	**36.21%**
Glue	62.8%	**66.08%**	**66.08%**
Hole P.	89.9%	90.33%	**95.29%**
Average	63.24%	64.4%	**73.64%**

TABLE C.1

C.2.1 Dataset by Krull [48]

Here we provide a table with detailed results on the dataset by Krull *et al.* [48]. In Table C.2, we show the percentage of correctly estimated poses for the different sequences, the combined results for the three different objects, and the average over the three objects. The results correspond to Figure 5.6. Again, the best results for each sequence and object are printed in bold numbers.

	Brachmann *et al.* [31]	*System II*
Cat 2	44.2%	**62.78%**
Samurai 2	33.7%	**60.47%**
Tool Box 1	**54.7%**	48.34%
Tool Box 2	**59.4%**	52.96%
Cat (total)	44.2%	**62.78%**
Samurai (total)	33.7%	**60.47%**
Tool Box (total)	**56.71%**	50.32%
Average	44.87%	**57.85%**

TABLE C.2

C.3 Details on the Calculation of Occlusion

The performance of the evaluated methods depends strongly on how much of the object is visible in the image and how much is occluded. To analyse this dependency, we calculated the percentage of occlusion for each object and image by rendering a depth image of the object in ground truth pose and doing a pixel-wise comparison to the recorded depth image. We count a pixel as occluded whenever the rendered depth at the pixel is more than 50 mm behind the recorded depth, or when there is no recorded depth value available for the pixel. To create Figure 5.5 we divided the test images into bins according to their level of occlusion and calculated the percentage of correctly estimated poses for each bin. We used a bin width of 10%.

C.4 Additional Qualitative Results

Here we provide qualitative results for four test cases. Two cases are taken from each dataset. In Figure C.1 and Figure C.2 we show results for the *can* and *cat* object, respectively. They belong to the dataset of [14] and [31] . In Figure C.3 and Figure C.4 we show results for the *samurai* and *tool box* object, respectively. They belong to the dataset from [48]. The upper four images in each figure show the RGB and depth channels as well as the forest predictions. The lower six are rendered and cropped images which are processed and fed into the CNN (see Figure 5.2(e-g)) to calculate an energy vale. Object coordinates in all figures were mapped to the RGB cube for visualization.

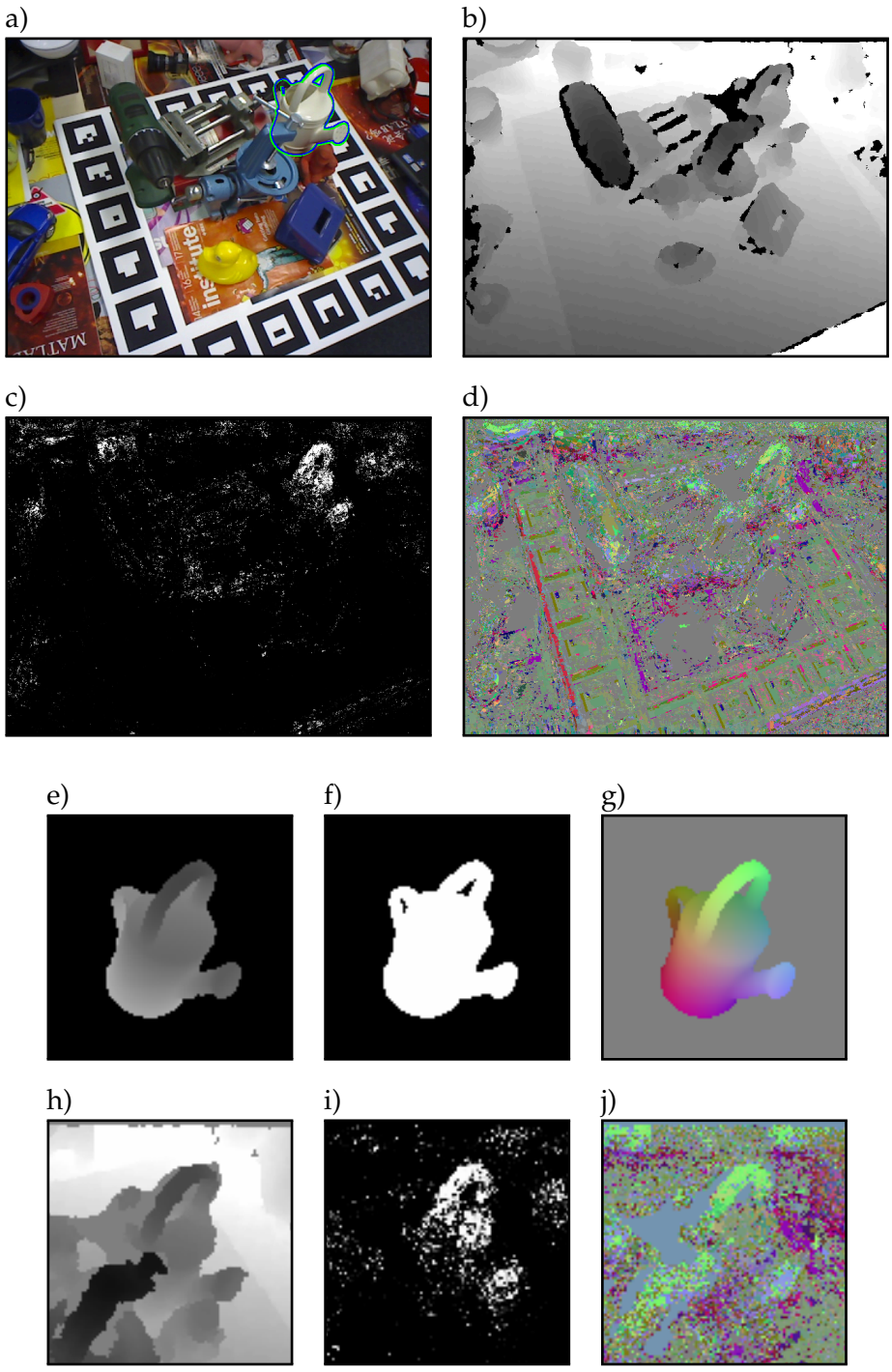


FIGURE C.1: Qualitative results. **a):** RGB image with results: blue indicates our pose estimate, green indicates the ground truth pose; **b):** recorded depth image; **c):** object probabilities predicted by the forest; **d):** object coordinates predicted by one tree from the forest; **e):** rendered depth image; **f):** rendered mask; **g):** rendered object coordinates; **h):** cropped observed depth image (values which are more than the object diameter behind or in front of the object are shown as white or black respectively); **i):** cropped object probabilities; **j):** cropped predicted object coordinates.

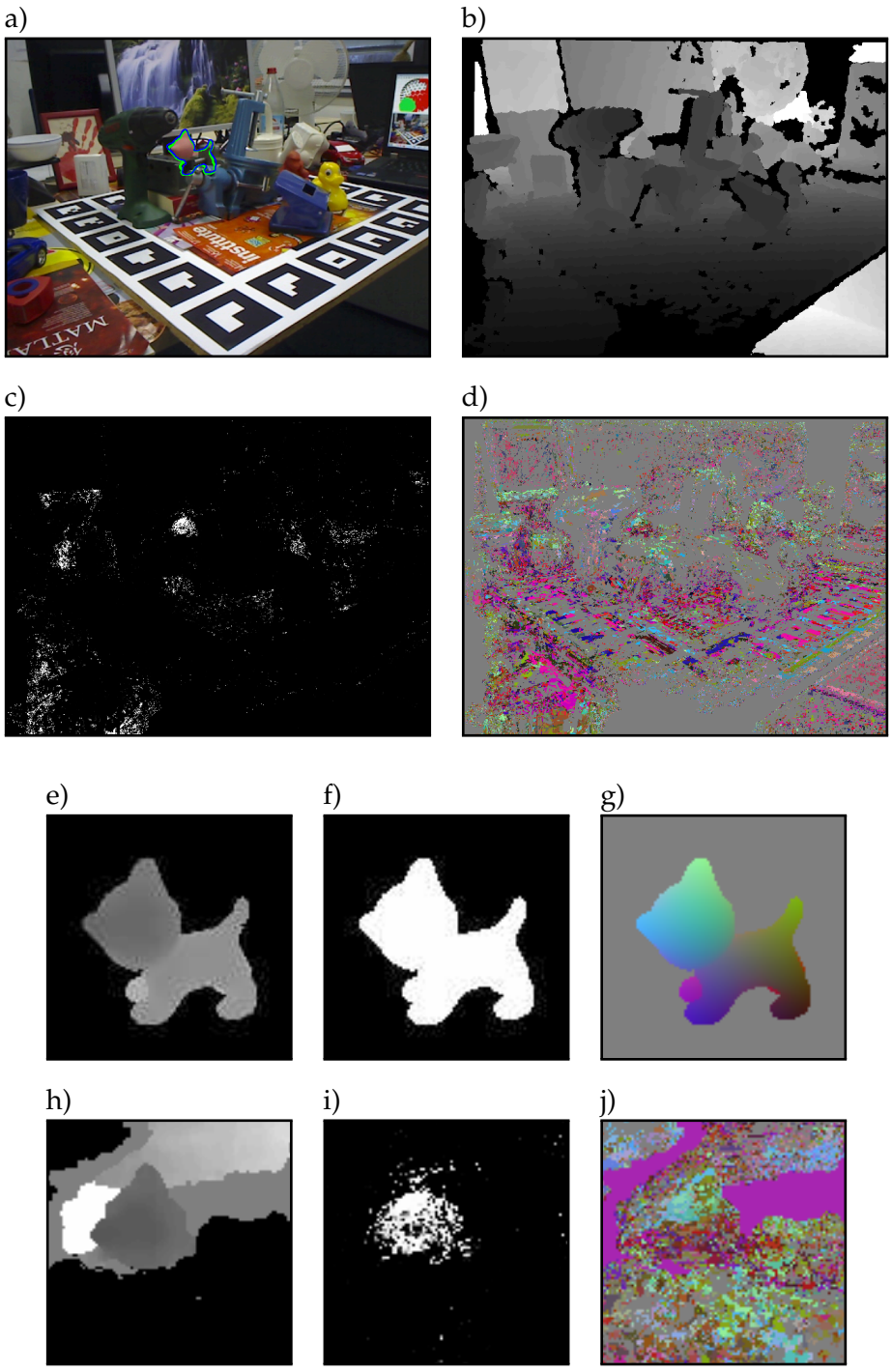


Figure includes images and models from the dataset by Hinterstoisser *et al.* [14], published in 2012 under a CC BY 4.0 Licence.

FIGURE C.2: Qualitative results. **a):** RGB image with results: blue indicates our pose estimate, green indicates the ground truth pose; **b):** recorded depth image; **c):** object probabilities predicted by the forest; **d):** object coordinates predicted by one tree from the forest; **e):** rendered depth image; **f):** rendered mask; **g):** rendered object coordinates; **h):** cropped observed depth image (values which are more than the object diameter behind or in front of the object are shown as white or black respectively); **i):** cropped object probabilities; **j):** cropped predicted object coordinates.

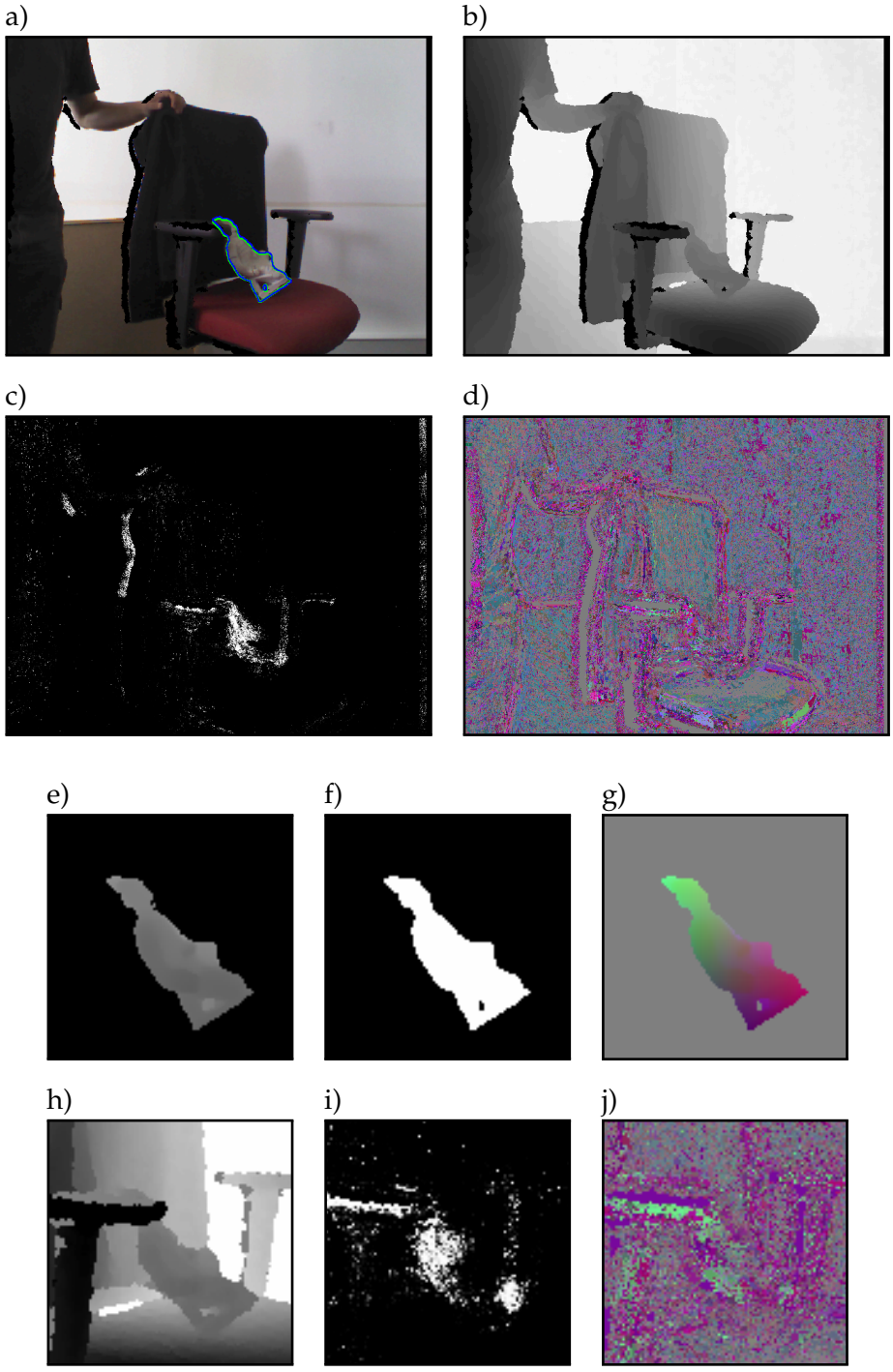


FIGURE C.3: Qualitative results. **a):** RGB image with results: blue indicates our pose estimate, green indicates the ground truth pose; **b):** recorded depth image; **c):** object probabilities predicted by the forest; **d):** object coordinates predicted by one tree from the forest; **e):** rendered depth image; **f):** rendered mask; **g):** rendered object coordinates; **h):** cropped observed depth image (values which are more than the object diameter behind or in front of the object are shown as white or black respectively); **i):** cropped object probabilities; **j):** cropped predicted object coordinates.

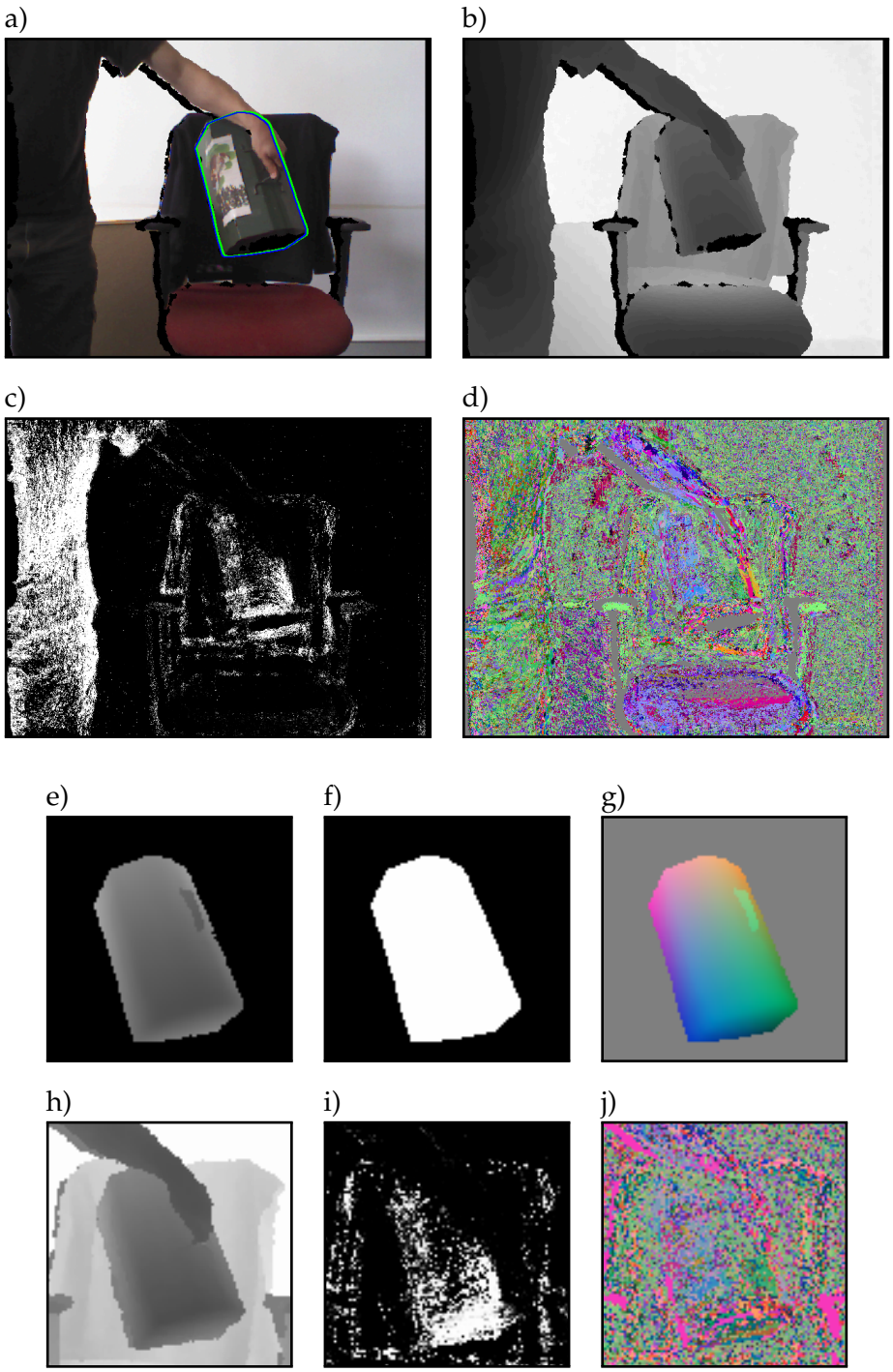


FIGURE C.4: Qualitative results. **a):** RGB image with results: blue indicates our pose estimate, green indicates the ground truth pose; **b):** recorded depth image; **c):** object probabilities predicted by the forest; **d):** object coordinates predicted by one tree from the forest; **e):** rendered depth image; **f):** rendered mask; **g):** rendered object coordinates; **h):** cropped observed depth image (values which are more than the object diameter behind or in front of the object are shown as white or black respectively); **i):** cropped object probabilities; **j):** cropped predicted object coordinates.

Appendix D

List of Abbreviations

2D, 3D, 6D	2-, 3-, 6-Dimensional
AR	Augmented Reality
CNN	Convolutional Neural Network
COBYLA	Constrained Optimization BY Linear Approximations
CRF	Conditional Random Field
HMM	Hidden Markov Model
MAP	Maximum A Posteriori
MCMC	Markov Chain Monte Carlo
PCL	Point Cloud Library
PF	Particle Filter
RANSAC	RAndom SAmple Consensus [54]
RGB	Red Green Blue
RGB-D	Dred Green Blue - Depth
RL	Reinforcement Learning
RMSE	Root Mean Square Error
SE(3)	Special Eucledian group
SIFT	Scale-Invariant Feature Transform [55]
SO(3)	Special Orthogonal group
SVD	Singular Value Decomposition
UARS	Uniform Axis Random Spin

Appendix E

List of Symbols

Here we provide a list of the used symbols. Symbols that are only used locally are not included.

Latin Symbols:

A	set of possible actions
$A(\cdot\|\cdot)$	acceptance probability
a	PoseAgent action
B^t	PoseAgent budget at time t
C	set of known objects
$C(\cdot)$	density of circular distribution
c	object of interest
D	2D table for accumulating derivatives
$D(a, \tau)$	table entry
$\boldsymbol{d}$	depth component of image $\boldsymbol{x}$
$\boldsymbol{d}_i$	3D camera coordinates of pixel i
d_i	1D depth information of pixel i
$\acute{\boldsymbol{d}}(H)$	rendered depth image using pose H
$\acute{\boldsymbol{d}}_i(H)$	rendered 3D camera coordinates of pixel i from $\acute{\boldsymbol{d}}$
$E(H)$	scoring function, measuring the quality of pose H
$E_{\mathrm{depth}}(H)$	depth component of scoring function $E(H)$
$E_{\mathrm{obj}}(H)$	object component of scoring function $E(H)$
$E_{\mathrm{coord}}(H)$	coordinate component of scoring function $E(H)$
$E(H; \boldsymbol{\theta})$	CNN-based energy function from *System II*
$E_a^t = E(s_a^t; \boldsymbol{\theta})$	PoseAgent energy for refinement
$E'^t_a = E'(s_a^t; \boldsymbol{\theta})$	PoseAgent energy for final decision
f	focal length of the camera
$\boldsymbol{f}_a^t$	additional features for a at time t
$f_{\mathrm{UARS}}(\cdot)$	density of UARS distribution
$f_{\mathrm{UARS-N}}(\cdot)$	density of UARS-Normal distribution
$f\left(\boldsymbol{d}_i, \acute{\boldsymbol{d}}_i(H)\right)$	distance measure on camera coordinates

$f_{\text{occ}}\left(\boldsymbol{d}_i, \acute{\boldsymbol{d}}_i(H)\right)$	modified distance measure on camera coordinates
G	1D table for accumulating derivatives
$G(j)$	table entry
$g\left(\boldsymbol{y}_{i,j}, \acute{\boldsymbol{y}}_i(H)\right)$	distance measure on object coordinates
H	pose hypothesis, $H = (R, \boldsymbol{v})$
$\hat{H}$	final pose estimate
H_t^k	particle k at time t
$\tilde{H}_t^k$, $\bar{H}_t^k$	intermediate particle k at time t
H_{t+1}^{est}	preliminary pose estimate
H_{t+1}^{local}	local preliminary pose estimate
H_{t+1}^{global}	global preliminary pose estimate
H^*	true pose of the object
$\boldsymbol{H}$	pose hypothesis pool
I	RGB component of image $\boldsymbol{x}$
i	pixel index
i_1, i_2, i_3	pixels indexes selected to generate a pose hypothesis
j	tree index
j_1, j_2, j_3	tree indexes selected to generate a pose hypothesis
K	number of samples/particles
k	index of sample/particle
L	set of training data
l	index of training iteration
$M_c(H)$	rendered silhouette of the object c under H
$M_c^L(H)$	like $M_c(H)$; additionally excluding pixels with low $p_{c,i}$
m^t	number of refinement steps done at time t
m_{max}	maximum number of allowed refinement steps
N	size of hypothesis pool
n	number of pixels in image $\boldsymbol{x}$
P_t	set of particles at time t
$\tilde{P}_t, \bar{P}_t$	set of intermediate particles at time t
$p_{c,i}^j$	predicted object probability for object c at pixel i by tree j
$p_{c,i}$	combined predicted object probability for object c at pixel i
$q(\cdot\|\cdot)$	proposal distribution
R	rotation matrix, element of $SO(3)$
R_0, R_1	rotation matrix, as base of tangent space
R_t^{pred}	predicted rotational component at time t
r	reward
$SO(3)$	Lie group of 3D rotations
$\mathfrak{so}(3)$	Lie algebra of $SO(3)$

S^t	PoseAgent state at time t
s_a^t	PoseAgent hyp. state of a at time t
s_a^τ	PoseAgent hyp. state after being chosen τ times
$\mathcal{T}$	random forest
$\|\mathcal{T}\|$	number of trees in random forest
T	last time index
t	time index
$\hat{\boldsymbol{u}}$	uniformly distributed 3D unit vector
$\boldsymbol{v}$	translation component of a pose
$\boldsymbol{v}_t^{\text{pred}}$	predicted translational component at time t
$\dot{\boldsymbol{v}}^t$	translational velocity at time t
$\dot{\boldsymbol{v}}_t^k$	translational velocity of particle k at time t
w_t^k	weight of particle k at time t
$\mathcal{X}_t$	state at time t
$\boldsymbol{x}$	input image, $\boldsymbol{x} = (\boldsymbol{d}, I)$
$\boldsymbol{y}$	image of object coordinates
$\boldsymbol{y}_{i,c}^j$	the object coordinates predicted for object c at pixel i by tree j
$\boldsymbol{y}_i$	3D object coordinates of pixel i
$\acute{\boldsymbol{y}}(H)$	rendered object coordinate image using pose H
$\acute{\boldsymbol{y}}_i(H)$	rendered object coordinates of pixel i from $\acute{\boldsymbol{y}}$
$\boldsymbol{z}$	observations including $\boldsymbol{x}$ and precomputed features
$\boldsymbol{z}_t$	observations at time t including $\boldsymbol{x}_t$ and precomputed features
$\acute{\boldsymbol{z}}(H)$	set of images, rendered using pose H

Greek Symbols:

α, β, γ	rotation angles
α^{prop}	parameter controlling the proposal distribution in *System I*
δ_c	diameter of the object c
$\boldsymbol{\theta}$	vector of CNN parameters
$\boldsymbol{\theta}^*$	vector of optimal CNN parameters
θ_j	individual CNN parameter with index j
κ	concentration parameter for circular distribution
κ^{prop}	concentration parameter of proposal distribution
λ_{depth}	weight of depth component in scoring function $E(H)$
λ_{obj}	weight of object component in scoring function $E(H)$
λ_{coord}	weight of coordinate component in scoring function $E(H)$
$\lambda_{\boldsymbol{v}}$	damping parameter for translation
λ_R	damping parameter for rotation
λ_{like}	parameter controlling harshness of obs. likelihood
λ^l	learning rate at iteration l
ν	learning rate decay parameter
$\pi\left(a^t \vert S^t; \boldsymbol{\theta}\right)$	PoseAgent policy
$\Sigma_{\boldsymbol{v}}$	translation covariance matrix for porp. dist. in *System II*
Σ_R	rotation covariance matrix for porp. dist. in *System II*
Σ^{mm}	covariance parameters of motion model in *System I*
Σ^{prop}	covariance parameters of proposal dist. in *System I*
$\sigma_R^{\text{mm}} = 1/\sqrt{\kappa^{\text{mm}}}$	standard deviation of rotation angles in motion model
$\sigma_R^{\text{prop}} = 1/\sqrt{\kappa^{\text{prop}}}$	standard deviation of rotation angles in proposal dist.
τ_d	threshold parameter, part of f
τ_d^{occ}	additional threshold parameter, part of f_{occ}
τ_y	threshold parameter, part of g
τ_a^t	number of times a has been chosen at time t
$\tau_{\max}$	maximum number of times hypothesis can be chosen
ψ	a rotation angle
$\phi(\psi)$	factor to compare densities in tangent spaces of $SO(3)$
$\psi_{R_0,R}$	angle of difference rotation between R_0 and R
$\boldsymbol{\omega}_\times$	element of $\mathfrak{so}(3)$, as skew symmetric matrix
$\boldsymbol{\omega} = (\omega^1, \omega^2, \omega^3)^\intercal$	3D Euler vector, corresponds to $\boldsymbol{\omega}_\times$
$\boldsymbol{\omega}_R$	3D vector in the tangent space at R
$\dot{\boldsymbol{\omega}}_t$	rotational velocity at time t
$\dot{\boldsymbol{\omega}}_t^k$	rotational velocity of particle k

Bibliography

[1]R. Eckhardt, "Stan Ulam, John von Neumann, and the Monte Carlo method", Los Alamos Science (1987).

[2]N. Metropolis and S. Ulam, *The Monte Carlo method*, 1949.

[3]N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines", The Journal of Chemical Physics **21**, 1087–1092 (1953).

[4]W. R. Gilks, S. Richardson, and D. Spiegelhalter, *Markov Chain Monte Carlo in practice* (CRC press, 1995).

[5]T.-P. Li and Y.-Q. Ma, "Analysis methods for results in gamma-ray astronomy", The Astrophysical Journal (1983).

[6]B. F. Manly, *Randomization, bootstrap and Monte Carlo methods in biology* (CRC Press, 2006).

[7]B. L. Hammond, W. A. Lester, and P. J. Reynolds, *Monte Carlo methods in ab initio quantum chemistry* (World Scientific, 1994).

[8]D. S. Oliver, L. B. Cunha, and A. C. Reynolds, "Markov Chain Monte Carlo methods for conditioning a permeability field to pressure data", Mathematical Geology (1997).

[9]R. Burch, F. N. Najm, P. Yang, and T. N. Trick, "A Monte Carlo approach for power estimation", IEEE Transactions on Very Large Scale Integration (VLSI) Systems (1993).

[10]W. Schneider, T. Bortfeld, and W. Schlegel, "Correlation between CT numbers and tissue parameters needed for Monte Carlo simulations of clinical dose distributions", Physics in medicine and biology (2000).

[11]P. Jäckel, *Monte Carlo methods in finance* (J. Wiley, 2002).

[12]U. Halekoh and W. Vach, "A Bayesian approach to seriation problems in archaeology", Computational statistics & data analysis (2004).

[13]R. M. Neal, "Probabilistic inference using Markov Chain Monte Carlo methods", (1993).

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

[15]P. Mamassian, "Prehension of objects oriented in three-dimensional space", Experimental Brain Research (1997).

[16]R. H. Cuijpers, J. B. Smeets, and E. Brenner, "On the relation between object shape and grasping kinematics", Journal of Neurophysiology (2004).

[17]M. A. Goodale and A. D. Milner, "Separate visual pathways for perception and action", Trends in neurosciences (1992).

[18]M. A. Goodale, "Transforming vision into action", Vision research (2011).

[19]M. A. Goodale, "How (and why) the visual control of action differs from visual perception", in Proc. r. soc. b (The Royal Society, 2014).

[20]C. McGinn, *Prehension: the hand and the emergence of humanity* (MIT Press, 2015).

[21]P. K. Janert, J. J. Shakes, N. M. Hanssens, and D. R. Hodge, *Time-based warehouse movement maps*, US Patent 7,243,001, 2007.

[22]E. H. Grosse, C. H. Glock, and W. P. Neumann, "Human factors in order picking: a content analysis of the literature", International Journal of Production Research (2017).

[23]Y. Uchiyama, K. Ebe, A. Kozato, T. Okada, and N. Sadato, "The neural substrates of driving at a safe distance: a functional MRI study", Neuroscience Letters (2003).

[24]X. S. Zheng and G. W. McConkie, "Two visual systems in monitoring of dynamic traffic: effects of visual disruption", Accident Analysis & Prevention (2010).

[25]N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Lessons from the amazon picking challenge", arXiv (2016).

[26]A. Edsinger and C. C. Kemp, "Human-robot interaction for cooperative manipulation: handing objects to one another", in Robot and human interactive communication, international symposium on (2007).

[27]C.-C. Wang, C. Thorpe, and A. Suppe, "Ladar-based detection and tracking of moving objects from a ground vehicle at high speeds", in Intelligent vehicles, symposium on (2003).

[28]C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking", The International Journal of Robotics Research (2007).

[29]D. I. Ferguson and D. Silver, *Pose estimation using long range features*, US Patent 9,062,979, 2015.

[30]H. Chen, A. S. Lee, M. Swift, and J. C. Tang, "3D collaboration method over HoloLens™ and Skype™ end points", in Proceedings of the 3rd international workshop on immersive media experiences (2015).

[31]E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6D object pose estimation using 3D object coordinates", in ECCV (2014).

[32]N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation", in Radar and signal processing (1993).

[33]R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction* (MIT press Cambridge, 1998).

[34]A. Tejani, D. Tang, R. Kouskouridas, and T.-K. Kim, "Latent-class hough forests for 3D object detection and pose estimation", in ECCV (2014).

[35]R. Rios-Cabrera and T. Tuytelaars, "Discriminatively trained templates for 3D object detection: a real time scalable approach", in CVPR (2013).

[36]Y. Park, V. Lepetit, and W. Woo, "Texture-less object tracking with online training using an RGB-D camera", in ISMAR (2011).

[37]C. Choi and H. I. Christensen, "Robust 3D visual tracking using particle filtering on the se (3) group", in ICRA (2011).

[38]C. Choi and H. I. Christensen, "3D textureless object detection and tracking: an edge-based approach", in IROS (2012).

[39]U. Asif, M. Bennamoun, and F. Sohel, "Real-time pose estimation of rigid objects using RGB-D imagery", in ICIEA (2013).

[40]J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection", in CVPR (2016).

[41]J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger", in CVPR (2017).

[42]T. Hodaň, X. Zabulis, M. Lourakis, Š. Obdržálek, and J. Matas, "Detection and fine 3D pose estimation of texture-less objects in RGB-D images", in IROS (2015).

[43]A. Kendall, M. Grimes, and R. Cipolla, "Posenet: a convolutional network for real-time 6-dof camera relocalization", in ICCV (2015).

[44]A. Kendall and R. Cipolla, "Modelling uncertainty in deep learning for camera relocalization", in ICRA (2016).

[45]E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image", in CVPR (2015).

[46]E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, and C. Rother, "DSAC - differentiable ransac for camera localization", in CVPR (2017).

[47]T. Hodaň, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, "T-LESS: an RGB-D dataset for 6D pose estimation of texture-less objects", in Wacv (2017).

[48]A. Krull, F. Michel, E. Brachmann, S. Gumhold, S. Ihrke, and C. Rother, "6-dof model based tracking via object coordinate regression", in ACCV (2014).

[49]T. Hodaň, J. Matas, and Š. Obdržálek, "On evaluation of 6d object pose estimation", in ECCV workshops (2016).

[50]B. Großmann, M. Siam, and V. Krüger, "Comparative evaluation of 3D pose estimation of industrial objects in RGB pointclouds", in International conference on computer vision systems (2015).

[51]K. McHenry, J. Ponce, and D. Forsyth, "Finding glass", in Computer vision and pattern recognition, 2005. cvpr 2005. ieee computer society conference on (2005).

[52]C. J. Phillips, M. Lecce, and K. Daniilidis, "Seeing glassware: from edge detection to pose estimation and shape recovery.", in Robotics: science and systems (2016).

[53]R. Feng and H. Zhang, "Efficient monocular coarse-to-fine object pose estimation", in ICMA (2016).

[54]M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", Communications of the ACM (1981).

[55]D. G. Lowe, "Object recognition from local scale-invariant features", in ICCV (1999).

[56]F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce, "3D object modeling and recognition using affine-invariant patches and multi-view spatial constraints", in CVPR (2003).

[57]I. Gordon and D. G. Lowe, "What and where: 3D object recognition with accurate pose", in *Toward category-level object recognition. lecture notes in computer science*, edited by J. Ponce, M. Hebert, C. Schmid, and A. Zisserman (Springer Berlin Heidelberg, Berlin, Heidelberg, 2006).

[58]A. C. Romea, D. Berenson, S. Srinivasa, and D. Ferguson, "Object recognition and full pose registration from a single image for robotic manipulation", in ICRA (2009).

[59]M. Martinez Torres, A. Collet Romea, and S. Srinivasa, "MOPED: a scalable and low latency object recognition and pose estimation system", in ICRA (2010).

[60]D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the hausdorff distance", TPAMI (1993).

[61]W. Kehl, F. Tombari, N. Navab, S. Ilic, and V. Lepetit, "Hashmod: a hashing method for scalable 3D object detection", in BMVC (2015).

[62]Y. Konishi, Y. Hanzawa, M. Kawade, and M. Hashimoto, "Fast 6D pose estimation from a monocular image using hierarchical pose trees", in ECCV (2016).

[63]P. Wohlhart and V. Lepetit, "Learning descriptors for object recognition and 3D pose estimation", in CVPR (2015).

[64]R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures", Communications of the ACM (1972).

[65]D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes", Pattern Recognition (1981).

[66]M. Sun, G. Bradski, B.-X. Xu, and S. Savarese, "Depth-encoded hough voting for joint object detection and shape recovery", in ECCV (2010).

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

[68]L. Breiman, "Random forests", Machine Learning **45**, 5–32 (2001).

[69]A. Doumanoglou, R. Kouskouridas, S. Malassiotis, and T.-K. Kim, "Recovering 6d object pose and predicting next-best-view in the crowd", in CVPR (2016).

[70]W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab, "Deep learning of local RGB-D patches for 3D object detection and 6d pose estimation", in ECCV (2016).

[71]B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: efficient and robust 3D object recognition.", in CVPR (2010).

[72]S. Hinterstoisser, V. Lepetit, N. Rajkumar, and K. Konolige, "Going further with point pair features", in ECCV (2016).

[73]J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon, "The vitruvian manifold: inferring dense correspondences for one-shot human pose estimation", in CVPR (2012).

[74]F. Michel, A. Krull, E. Brachmann, M. Y. Yang, S. Gumhold, and C. Rother, "Pose estimation of kinematic chain instances via object coordinate regression", in CVPR (2015).

[75]J. Mund, F. Michel, F. Dieke-Meier, H. Fricke, L. Meyer, and C. Rother, "Introducing lidar point cloud-based object classification for safer apron operations", in ESAVS (2016).

[76]A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning", in CVPR (2017).

[77]S. Mahendran, H. Ali, and R. Vidal, "3D pose regression using convolutional neural networks", in ICCV workshops (2017).

[78]A. Doumanoglou, V. Balntas, R. Kouskouridas, and T.-K. Kim, "Siamese regression networks with efficient mid-level feature extraction for 3D object pose estimation", in CoRR (2016).

[79]Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: a convolutional neural network for 6D object pose estimation in cluttered scenes", arXiv (2017).

[80]R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", Machine learning (1992).

[81]A. Krull, E. Brachmann, F. Michel, M. Ying Yang, S. Gumhold, and C. Rother, "Learning analysis-by-synthesis for 6D pose estimation in RGB-D images", in ICCV (2015).

[82]A. Krull, E. Brachmann, S. Nowozin, F. Michel, J. Shotton, and C. Rother, "PoseAgent: budget-constrained 6d object pose estimation via reinforcement learning", in CVPR (2017).

[83]J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. W. Fitzgibbon, "Scene coordinate regression forests for camera relocalization in RGB-D images", in CVPR (2013).

[84]A. Criminisi and J. Shotton, *Decision forests for computer vision and medical image analysis* (Springer Science & Business Media, 2013).

[85]Y. Cheng, "Mean shift, mode seeking, and clustering", TPAMI (1995).

[86]W. Kabsch, "A solution for the best rotation to relate two sets of vectors", Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography (1976).

[87]J. Gallier and J. Quaintance, *Notes on differential geometry and Lie groups*, 2017.

[88]R. Tron, R. Vidal, and A. Terzis, "Distributed pose averaging in camera networks via consensus on se (3)", in ICDSC (2008).

[89]J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors", Matrix (2006).

[90]M. D. Shuster, "A survey of attitude representations", Navigation (1993).

[91]M. A. Bingham, D. J. Nordman, and S. B. Vardeman, "Modeling and inference for measured crystal orientations and a tractable class of symmetric distributions for rotations in three dimensions", Journal of the American Statistical Association (2009).

[92]E. Eade, *Lie groups for 2D and 3D transformations*, 2013.

[93]S. Bosch, *Algebra*, Springer-Lehrbuch (Springer Berlin Heidelberg, 2009).

[94]J. M. Lee, *Smooth manifolds* (Springer, 2003).

[95]R. Hartley, J. Trumpf, Y. Dai, and H. Li, "Rotation averaging", International journal of computer vision (2013).

[96]J. Milnor, *Geometry*, Collected Papers Series (Publish or Perish, 1994).

[97]R. I. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, Second edition (Cambridge University Press, 2004).

[98]K. Wilson, D. Bindel, and N. Snavely, "When is rotations averaging hard?", in ECCV (2016).

[99]M. Moakher, "Means and averaging in the group of rotations", SIAM journal on matrix analysis and applications (2002).

[100]C. Gramkow, "On averaging rotations", Journal of Mathematical Imaging and Vision (2001).

[101]G. Klein and D. W. Murray, "Full-3D edge tracking with a particle filter", in BMVC (2006).

[102]P. Azad, D. Munch, T. Asfour, and R. Dillmann, "6-DoF model-based tracking of arbitrarily shaped 3D objects", in ICRA (2011).

[103]C. A. León, J.-C. Massé, and L.-P. Rivest, "A statistical model for random rotations", Journal of Multivariate Analysis (2006).

[104]J. Kwon, M. Choi, F. C. Park, and C. Chun, "Particle filtering on the Euclidean group: framework and applications", Robotica **25**, 725–737 (2007).

[105]Y. Qiu, D. J. Nordman, and S. B. Vardeman, "A wrapped trivariate normal distribution and Bayes inference for 3D rotations", Statistica Sinica (2014).

[106]A. Haar, "Der massbegriff in der theorie der kontinuierlichen gruppen", Annals of Mathematics **34**, 147–169 (1933).

[107]D. Vvedensky, *Lecture notes on group theory*, 2001.

[108]T. D. Downs, "Orientation statistics", Biometrika (1972).

[109]C. Khatri and K. Mardia, "The von Mises-Fisher matrix distribution in orientation statistics", Journal of the Royal Statistical Society. Series B (Methodological) (1977).

[110]H. Bunge, *Texture analysis in materials science: mathematical methods* (Butterworths, 1982).

[111]S Matthies, J Muller, and G. Vinel, "On the normal distribution in the orientation space", Texture, Stress, and Microstructure (1988).

[112]D. I. Nikolayev and T. I. Savyolov, "Normal distribution on the rotation group so (3)", Texture, Stress, and Microstructure (1997).

[113]K. V. Mardia and P. E. Jupp, *Directional statistics* (John Wiley & Sons, 2009).

[114]D Collett and T Lewis, "Discriminating between the von Mises and wrapped normal distributions", Australian & New Zealand Journal of Statistics (1981).

[115]B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning", in CVPR (2009).

[116]S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. L. Hicks, and P. H. Torr, "Struck: structured output tracking with kernels", TPAMI (2016).

[117]Z. Chen, "Bayesian filtering: from Kalman filters to particle filters, and beyond", Statistics (2003).

[118]R. E. Kalman, "A new approach to linear filtering and prediction problems", Journal of Basic Engineering (1960).

[119]S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems", in Int. symp. aerospace/defense sensing, simul. and controls (1997).

[120]M. Isard and A. Blake, "Contour tracking by stochastic propagation of conditional density", in ECCV (1996).

[121]M. Pupilli and A. Calway, "Real-time camera tracking using known 3D models and a particle filter", in ICPR (2006).

[122]M. Bray, E. Koller-Meier, and L. Van Gool, "Smart particle filtering for 3D hand tracking", in Automatic face and gesture recognition, international conference on (2004).

[123]C. Teuliere, E. Marchand, and L. Eck, "Using multiple hypothesis in model-based tracking", in ICRA (2010).

[124]M. McElhone, J. Stuckler, and S. Behnke, "Joint detection and pose tracking of multi-resolution surfel models in RGB-D", in ECMR (2013).

[125]A. Chiuso and S. Soatto, "Monte Carlo filtering on Lie groups", in Conference on Decision and Control (2000).

[126]J. Stückler and S. Behnke, "Multi-resolution surfel maps for efficient dense 3D modeling and tracking", en, Journal of Visual Communication and Image Representation **25**, 137–147 (2014).

[127]C. Choi and H. I. Christensen, "RGB-D object tracking: a particle filter approach on gpu", in IROS (2013).

[128]L. Rabiner and B Juang, "An introduction to hidden markov models", ASSP Magazine (1986).

[129]A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering", Statistics and Computing (2000).

[130]M. J. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation", in Advances in Optimization and Numerical Analysis (1994).

[131]S. Johnson, *The nlopt nonlinear-optimization package*.

[132]G. Fanelli, J. Gall, and L. Van Gool, "Real time head pose estimation with random regression forests", in CVPR (2011).

[133]S. Song and J. Xiao, "Tracking revisited using RGBD camera: unified benchmark and baselines", in ICCV (2013).

[134]C. Bersch, D. Pangercic, S. Osentoski, K. Hausman, Z.-C. Marton, R. Ueda, K. Okada, and M. Beetz, "Segmentation of textured and textureless objects through interactive perception", in Rss workshop on robots in clutter: manipulation, perception and navigation in human environments (2012).

[135]M. Hejrati and D. Ramanan, "Analysis by synthesis: 3D object recognition by object reconstruction", in CVPR (2014).

[136]P. Isola and C. Liu, "Scene collaging: analysis and synthesis of natural images with semantic layers", in ICCV (2013).

[137]J. Gall, B. Rosenhahn, and H. Seidel, "Drift-free tracking of rigid and articulated objects", in CVPR (2008).

[138]A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in NIPS (2012).

[139]N. Zhang, J. Donahue, R. B. Girshick, and T. Darrell, "Part-based R-CNNs for fine-grained category detection", in ECCV (2014).

[140]P. Agrawal, R. B. Girshick, and J. Malik, "Analyzing the performance of multilayer neural networks for object recognition", in ECCV (2014).

[141]M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks", in CVPR (2014).

[142]J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation", in CVPR (2015).

[143]A. Toshev and C. Szegedy, "DeepPose: human pose estimation via deep neural networks", in CVPR (2014).

[144]S. Gupta, R. Girshick, P. Arbelaez, and J. Malik, "Learning rich features from RGB-D images for object detection and segmentation", in ECCV (2014).

[145]A.Dosovitskiy, J.T.Springenberg, and T.Brox, "Learning to generate chairs with convolutional neural networks", in CVPR (2015).

[146]S. Gupta, P. Arbeláez, R. Girshick, and J. Malik, "Inferring 3D object pose in RGB-D images", in CVPR (2015).

[147]S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification", in CVPR (2005).

[148]J. Zbontar and Y. LeCun, "Computing the stereo matching cost with a convolutional neural network", CoRR (2014).

[149]M. Weinmann, J. Gall, and R. Klein, "Material classification based on training data synthesized using a BTF database", in ECCV (2014).

[150]Y. Sugano, Y. Matsushita, and Y. Sato, "Learning-by-synthesis for appearance-based 3D gaze estimation", in CVPR (2014).

[151]T. Kulkarni, I. Yildirim, W. Freiwald, and J. Tenenbaum, "Deep generative vision as approximate Bayesian computation", in NIPS workshops (2014).

[152]J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: probabilistic models for segmenting and labeling sequence data", in ICML (2001).

[153]L. Bottou, "Stochastic gradient learning in neural networks", in Proceedings of Neuro-Nımes (1991).

[154]R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation.", in NIPS (1999).

[155]E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning", Journal of Machine Learning Research (2004).

[156]W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots", in ICRA (2002).

[157]P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight", in NIPS (2007).

[158]E. M. Schwartz, E. T. Bradlow, and P. S. Fader, "Customer acquisition via display advertising using multi-armed bandit experiments", Marketing Science (2017).

[159]A. Ghaffari, "Real-time routing algorithm for mobile ad hoc networks using reinforcement learning and heuristic algorithms", Wireless Networks (2017).

[160]V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", Nature (2015).

[161]O. Teboul, I. Kokkinos, L. Simon, P. Koutsourakis, and N. Paragios, "Shape grammar parsing via reinforcement learning", in CVPR (2011).

[162]V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention", in NIPS (2014).

[163]J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning", in CVPR (2015).

[164]S. Mathe, A. Pirinen, and C. Sminchisescu, "Reinforcement learning for visual object detection", in CVPR (2016).

[165]J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention", in ICLR (2015).

[166]L. Bottou, "Stochastic gradient tricks", in *Neural networks, tricks of the trade, reloaded* (2012).

[167]E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models", arXiv (2015).

[168]M. McGill and P. Perona, "Deciding how to decide: dynamic routing in artificial neural networks", arXiv (2017).

[169]V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang, "Skip RNN: learning to skip state updates in recurrent neural networks", arXiv (2017).

[170]A. Kulesza and B. Taskar, "Determinantal point processes for machine learning", Foundations and Trends in Machine Learning (2012).

[171]A. Kirillov, B. Savchynskyy, D. Schlesinger, D. Vetrov, and C. Rother, "Inferring m-best diverse labelings in a single one", in ICCV (2015).

[172]R. Kumar and D. Batra, "Pose tracking by efficiently exploiting global features", in WACV (2016).

[173]M. Meila, *Lecture notes on statistical learning: modeling, prediction and computing*, 2012.

[174]S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks", in NIPS (2015).

[175]J. Johnson, A. Karpathy, and L. Fei-Fei, "Densecap: fully convolutional localization networks for dense captioning", in ICCV (2016).

[176]J. Domke, "Learning graphical model parameters with approximate marginal inference", TPAMI (2013).

[177]Z. Liu, X. Li, P. Luo, C.-C. Loy, and X. Tang, "Semantic image segmentation via deep parsing network", in CVPR (2015).

[178]S. Ross, D. Munoz, M. Hebert, and J. A. Bagnell, "Learning message-passing inference machines for structured prediction", in CVPR (2011).

[179]P. Abbeel, A. Coates, M. Montemerlo, A. Y. Ng, and S. Thrun, "Discriminative training of Kalman filters.", in Robotics: Science and Systems (2005).

[180]R. Hess and A. Fern, "Discriminatively trained particle filters for complex multi-object tracking", in CVPR (2009).

[181]M. C. Burkhart, D. M. Brandman, C. E. Vargas-Irwin, and M. T. Harrison, "The discriminative Kalman filter for nonlinear and non-Gaussian sequential Bayesian filtering", arXiv (2016).

[182]M. Taboga, *Lectures on probability theory and mathematical statistics* (CreateSpace Independent Pub., 2012).

[183]S. Sra, "A short note on parameter approximation for von Mises-Fisher distributions: and a fast implementation of I s (x)", Computational Statistics (2012).