Dipl.-Ing. Markus Oberweger

3D Hand Pose Estimation from Images for Interactive Applications

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor
Prof. Dr. Vincent Lepetit
Institute for Computer Graphics and Vision, Graz University of Technology

Reviewer
Prof. Dr. Christian Theobalt
Max-Planck-Institute for Informatics

Graz, Austria, Nov. 2017

To my beloved ones

We are stuck with technology when what we really want is just stuff that works.

Douglas Adams, The Salmon of Doubt (2002)

Abstract

Hands are the most important body part for humans to interact with and manipulate their environment. Hence, 3D hand pose estimation is an important cornerstone of many Human-Computer Interaction (HCI), Virtual Reality (VR), and Augmented Reality (AR) applications, such as robotic control or virtual object interaction. Despite the plentiful applications and importance for the industry, hand pose estimation is still an unsolved problem. It requires dealing with many challenges, including many Degrees of Freedom (DoF) and occlusions.

In this thesis we focus on markerless Computer Vision-based 3D hand pose estimation as well as joint hand-object pose estimation in case a hand is interacting with an object. The input to our methods are single view depth or color images and the outputs are the 3D poses of the hand and/or the object, respectively.

First, we contribute to the class of discriminative methods for 3D hand pose estimation. Our proposed methods are based on Convolutional Neural Networks (CNNs), and we show how to efficiently integrate a prior on the 3D pose into such a network. The prior significantly increases the accuracy of the predicted 3D hand pose and restricts the predicted pose to the manifold of feasible hand poses. Despite the high accuracy, the pose estimator can still make mistakes and thus we propose a second approach that learns to fix the mistakes made by an initial estimator. This approach relies on the analysis-by-synthesis paradigm to fix these mistakes.

However, these supervised methods require a large amount of accurate training data, which can be cumbersome to acquire. Therefore, we propose an efficient method to generate 3D training data for articulated objects. Our semi-automatic method takes image sequences with sparse 2D annotations as input and outputs 3D annotations for every frame. We also investigate how additional data sources can be leveraged. We present an approach that additionally uses synthetically rendered depth images of the hand, which are easy to generate in practice, and using this additional data source significantly boosts the accuracy of the predictions. We further investigate 3D hand pose estimation from color images.

Therefore, we propose a method that transfers the supervision of pairs of unlabeled depth and color images to labeled synthetic depth images. We show that our method—trained without color labels—can achieve similar results to methods trained on labeled color images.
Practical applications in *AR* and *VR* often involve hand-held objects that are manipulated by the hand. These applications require the pose of the hand together with the pose of the object, and thus we introduce an approach that jointly estimates the hand and the object pose. Our method is, to the best of our knowledge, the first single-frame method for joint hand-object pose estimation, and has a comparable accuracy to state-of-the-art tracking-based approaches. When the hand interacts with an object, handling the different occlusions quickly becomes intractable and methods that do not need any prior knowledge of the occluders are required. For such scenarios, we introduce a method for 3D object pose estimation that is inherently robust to occlusions and does not require knowledge of the occluders in advance, while providing high accuracy 3D pose estimates of highly occluded objects.
Finally, we conclude with a discussion of the proposed methods and possible extensions with directions of future work.

Keywords. 3D hand pose estimation; 3D object pose estimation; Joint hand-object pose estimation; Deep Learning; Occlusions

Kurzfassung

Hände sind der wichtigste Körperteil mit dem Menschen mit ihrer Umwelt interagieren und diese manipulieren. Daher ist die 3D Handposenschätzung ein integraler Bestandteil vieler Anwendungen im Bereich der Mensch-Computer-Interaktion (HCI), virtuellen Realität (VR) und erweiterten Realität (AR), wie zum Beispiel die Steuerung von Robotern oder Interaktion mit virtuellen Objekten. Trotz der zahlreichen Anwendungen und der großen Bedeutung für die Industrie ist die Handposenschätzung immer noch ein ungelöstes Problem. Es erfordert den Umgang mit vielen Herausforderungen, darunter einer hohen Anzahl an Freiheitsgraden und Verdeckungen.

In dieser Dissertation fokussieren wir uns auf die markerlose 3D Handposenschätzung mithilfe maschineller Bilderverarbeitung, sowie auf die Schätzung der gemeinsamen Hand- und Objektpose für den Fall, dass eine Hand mit einem Objekt interagiert. Die Eingabedaten für die präsentierten Methoden sind Einzelbilder mit Tiefen- oder Farbinformation und die Ausgabendaten sind die 3D Posen der Hand bzw. des Objekts.

Zu Beginn präsentieren wir diskriminative Methoden für die 3D Handposenschätzung. Unsere Methoden basieren auf künstlichen neuronalen Netzwerken und wir zeigen, wie man einen Prior der 3D Handpose effizient in ein solches Netzwerk integriert. Dieser Prior erhöht signifikant die Genauigkeit der vorhergesagten 3D Handposen und beschränkt diese auf die Mannigfaltigkeit möglicher Posen. Trotz der hohen Genauigkeit kann der Posenschätzer immer noch Fehler machen und so präsentieren wir eine weitere Methode, welche lernt, die Fehler, die von einem solchen Schätzer gemacht wurden, zu korrigieren. Diese Methode beruht auf dem Analyse-durch-Synthese Paradigma.

Diese Methoden erfordern jedoch eine große Anzahl an Trainingsdaten, welche mühsam erstellt werden müssen. Wir schlagen daher eine effiziente Methode vor, um 3D Trainingsdaten für artikulierte Objekte zu generieren. Unsere halbautomatische Methode verwendet Bildsequenzen mit vereinzelten 2D Annotationen und erzeugt daraus 3D Annotationen für jedes Einzelbild. Weiters untersuchen wir, wie zusätzliche

Datenquellen genutzt werden können. Wir präsentieren einen Ansatz, der zusätzlich synthetisch generierte Tiefenbilder einer Hand verwendet, welche in der Praxis einfach zu erstellen sind. Durch die Verwendung dieser zusätzlichen Datenquelle können wir die Genauigkeit der Vorhersagen signifikant erhöhen. Wir untersuchen außerdem die 3D Handposenschätzung aus Farbbildern. Dazu präsentieren wir eine Methode, welche die Information, die in der Korrespondenz von Paaren von Tiefen- und Farbbildern steckt, auf annotierte synthetische Tiefenbilder überträgt. Unsere Methode, ohne annotierte Farbbilder trainiert, erzielt ähnliche Ergebnisse wie Methoden, die mit annotierten Farbbildern trainiert wurden.
Praktische Anwendungen in Bereich der AR und VR beinhalten oft handgehaltene Objekte, welche von der Hand manipuliert werden. Diese Anwendungen erfordern die Schätzung der Handpose gemeinsam mit der Objektpose. Wir stellen daher eine solche Methode vor, die gemeinsam die Hand- und Objektpose schätzt. Unsere Methode ist nach unserem besten Wissen die erste Methode zur gemeinsamen Schätzung von Hand- und Objektpose aus einem Einzelbild und bietet eine vergleichbare Genauigkeit wie Tracking-basierte Ansätze. Wenn die Hand mit einem Objekt interagiert, wird die Handhabung der verschiedenen Verdeckungen schnell komplex und Methoden, die keine vorherige Kenntnis der Verdecker benötigen, sind erforderlich. Für solche Szenarien präsentieren wir eine Methode zur Schätzung der 3D Pose, die inhärent robust gegenüber Verdeckungen ist und keine vorherige Kenntnis der Verdecker erfordert. Unsere Methode bietet eine sehr hohe Genauigkeit für die 3D Posenschätzung stark verdeckter Objekte.
Abschließend folgt eine Diskussion der vorgeschlagenen Methoden und mögliche Erweiterungen und Richtungen für zukünftige Arbeiten.

Schlagwörter. 3D Handposenschätzung; 3D Objektposenschätzung; Gemeinsame Hand- und Objektposenschätzung; Deep Learning; Verdeckungen

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature

Acknowledgments

At the end of my PhD studies, there comes the time to thank all the people who contributed to this thesis in various ways and without their help coming to this point might not have even been possible.
First and foremost, my sincere gratitude goes to Prof. Vincent Lepetit, who gave me the opportunity to start in his, back then, newly founded lab, and for his support during my studies. He was a great supervisor, taking his time for interesting discussions, providing guidance and brave new ideas, and giving me the freedom to peruse own ideas. I am also very grateful to Prof. Christian Theobalt for taking his precious time and acting as a reviewer for this thesis.
I would also like to thank the alumni of the ICG, especially Michael, Andreas, and Stefan, for sparking the interest in scientific research during my Master studies and encouraging me to start a PhD study, ultimately resulting in this thesis.
Special thanks goes to all members of the Computer Vision for Augmented Reality group. Most importantly I would like to thank my office-mate Mahdi, for the interesting discussions, long deadline nights, and the fruitful collaborations. My gratitude goes also to Paul for taking me under his wings, Peter M. for his seemingly endless support for not only regulatory questions, Anil a fellow teammate from the very first moment, Martin, Alex, Sinisa, and Mina. Carmen also deserves my gratitude for her altruistic support.
I also want to mention the folks from the reading group for calibrating my paper reading skills by motivating me to read a new paper every week.
Generally, I have to thank everybody at the ICG for chats, discussions, joint teaching efforts, company during lunch and coffee breaks, parties, exciting conference visits, and especially for creating this great place to work.
Another big thanks goes to my students, Markus and Samuel, which was a pleasure to supervise and for their hard and fruitful work.
I would also like to thank the folks from Facebook Reality Labs, formerly Oculus Re-

search, formerly NimbleVR, especially Rob, Chris, Kenrick, Yuting, Po-Chen, Wenhao, Shangchen, Zeyuan, for the awesome internship and collaboration.
Last and most importantly, I would like to thank my family, especially my parents Rosi and Helmut, and my brother Stefan, for supporting my studies and decisions, motivating me to pursue my goals, and making my limited leisure time as enjoyable as possible.

This thesis is based upon work supported by the Christian Doppler Laboratory for Handheld Augmented Reality, the Graz University of Technology FutureLabs fund, the Christian Doppler Laboratory for Semantic 3D Computer Vision, Oculus Research, and Qualcomm.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Problem Statement

Accurate 3D hand pose estimation is an important requirement for many Human-Computer Interaction (HCI), Virtual Reality (VR), and Augmented Reality (AR) applications. It has attracted lots of attention in the Computer Vision research community [122, 165, 187, 201, 256, 299] with works reaching back to the early 1990s [108, 209]. In this thesis, we aim to advance the state-of-the-art in 3D hand pose estimation and joint hand-object pose estimation. Specifically, we want to estimate the joint locations of a human hand from an input image. Early works used 2D images [108, 209], which renders this problem very difficult due to ambiguities in the 2D images and given the restricted computational power at that time. With the advent of 3D sensors, such as structured-light or time-of-flight sensors, inferring 3D joint locations became feasible. Using 3D measurements as input to 3D pose estimation methods makes pose prediction tractable and one of the seminal works in this field used the Microsoft Kinect, a structured-light imaging sensor, for 3D human pose estimation [229]. The Kinect was designed for long range sensing but the introduction of short range sensors, such as the Creative Senz3D or Intel RealSense sensors, gave rise to accurate 3D hand

pose estimation from depth images [122, 165, 256]. More recently, hand pose estimation from color images became feasibly [172, 325], by using large amounts of training data to resolve the ambiguities present in the 2D images. Industry also developed task-tailored devices such as the Leap Motion for 3D hand pose estimation, however, they still lack robustness and accuracy required for many real world applications.

In this work we focus on 3D hand pose *regression*, which is contrary to gesture recognition [207] that aims at assigning a distinctive label such as `open` or `close` to a certain hand pose. In regression we deal with continuous labels in Euclidean space that assign a location in $\mathbb{R}^3$ to each joint. The term *3D* indicates that the output of our methods is in 3D space relative to the camera, which is contrary to other works that only perform 2D pose estimation [179, 290], *i.e.*, estimating the locations in 2D image coordinates. The input data we use can be from 3D sensors as well as 2D color images. Our methods only require a *single* camera view, which is very challenging as there is no simple way to resolve the inherent ambiguities of hand pose estimation. We perform *markerless* pose estimation, which implies that we use the users' bare hands as input without invasive markers or other elements added to the users' hands. While our methods were initially proposed for hands in isolation, we extend them to jointly estimate the pose of the hand and an object when both are interacting with each other. Our methods are computationally very efficient as they run in *real-time* on consumer hardware. This efficiency is an important requirement for interactive applications and thus gives rise to plentiful future applications.

The human hand representation that is commonly used in the literature [149] is shown in Figure 1.1. Each finger consists of three joints: the metacarpophalangeal (MCP), proximal interphalangeal (PIP), and distal interphalangeal (DIP) joint. Each joint is assigned a specific number of Degrees of Freedom (DoF), depending on the physiology of the joint. The wrist has six *DoF* as it can move freely around the body comprising 3D translation and global rotation. The MCP joints have two *DoF*, PIP and DIP joints have one *DoF*. The thumb has a slightly different physiology than the other fingers, consisting of the carpometacarpal (CMC) joint with two *DoF*, the MCP joint with two *DoF* and the interphalangeal (IP) joint with one *DoF*. This sums up to a total of 27 *DoF*.[1]

In a more formal definition of 3D pose estimation, we aim at mapping an input image to the corresponding 3D hand pose $\mathbf{P}$. As input we consider depth $\mathcal{D}$ or color images I. This mapping task does not significantly differ from other pose estimation problems, such as human pose [229]. However, hand pose estimation has unique challenges as we discuss in Section 1.3. On of the biggest challenges is the high number of *DoF* due to the complex physiology of the human hand. In this thesis, the hand pose is represented by its J 3D hand joint locations $\mathbf{P} = \{\mathbf{j}_i\}_{i=1}^{J}$, *i.e.*, each joint $\mathbf{j}_i$ is assigned $\mathbf{j}_i = (x_i, y_i, z_i)$ coordinates

[1]The representation of the hand model is not limited to the one we use in this work, since related works use other models resulting in different *DoF*, *e.g.*, [2] use one additional *DoF* for the MCP and CMC joints to represent more natural movements, or [271] use a Linear Blend Skinning (LBS) model with 42 *DoF*. However, since our methods are all learning-based, they can be trained to predict other hand model representations as well.

in the camera coordinate system. Given a hand model with bone length and rotation angles for all *DoF* of the joints, forward kinematics [86] can be used to calculate the 3D joint locations. Reciprocally, inverse kinematics [268] can be applied to obtain joint angles and bone length from the 3D joint locations.

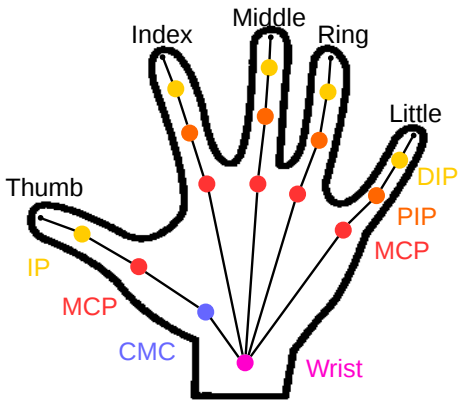


Figure 1.1: Human hand model with joints. Each joint is assigned a number of *DoF*: MCP and CMC joints have two *DoF*, IP, PIP and DIP joints have one *DoF*. The global hand location and orientation is defined by the wrist with six *DoF*.

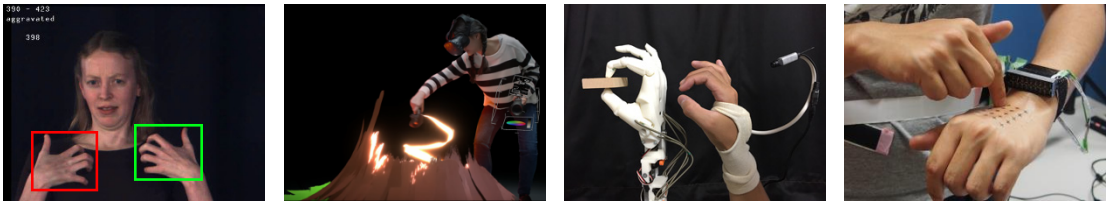
1.2 Applications

There are many applications of 3D hand pose estimation in *AR*, *VR*, and *HCI*. These applications require the pose estimation methods be robust to occlusions, hand size and shape, background clutter, and noise. Further, they require a high accuracy, low latency, and high speed for interactive applications. Especially on mobile devices, a low computational complexity is desired in order to run on lightweight mobile hardware, such as the Nvidia Tegra[2].

Humans use their hands as primary body part to interact with their environment, which naturally gives rise to plentiful applications within *HCI* as depicted in Figure 1.2. Such applications include sign language recognition [176], where an algorithm automatically assigns the corresponding sign language entity to a specific hand pose. Another application includes in-air drawing [77], where a user draws sketches mid-air and a sensor captures the trajectory. Capturing the human hand pose can be used for robotic control [245], where a robotic hand mimics the hand pose of a human operator. Touch-based user interfaces [109] can also be realized by capturing the users hand pose and detecting commands.

Further, 3D hand pose estimation is a prerequisite of many *AR* and *VR* applications and the users' hands can be used as a primary tool for interacting with virtual elements, as

[2] `https://www.nvidia.com/object/tegra.html`



(a) Sign language recognition. **(b)** In-air drawing. **(c)** Robotic control. **(d)** Touch interfaces.

Figure 1.2: Applications of hand pose estimation within *HCI*, including sign language recognition [176], in-air drawing [77], robotic control [245], and touch-based user interfaces [109].

shown in Figure 1.3. A user should be able to affect the virtual world with, for example, pushes or grabs, or interact with virtual interfaces using gestures. [98] has shown an interaction method in *VR*, where the bare hand can be used to interact with virtual objects, or the hand pose is captured and virtual objects are superimposed on the hand in an *AR* environment [21]. Further, the hand pose can be used to control virtual interfaces [9] for example in an automotive environment. The video gaming industry is also highly interested in using the 3D hand pose for interaction in video games [99], where, so far, additional hardware is required in order to provide the interfaces. Especially in *VR*, self presence is an important prerequisite for an immersive experience, *i.e.*, the user can see its own body and hands that need to be accurately modeled, and incorporating hands into social avatars is an important part for increased immersiveness [19].

(a) Object manipulation in *AR*. **(b)** Object interaction in *VR*. **(c)** Virtual interfaces. **(d)** Video games.

Figure 1.3: Applications of hand pose estimation in *AR* and *VR*, including object manipulation in *AR* [21], object interaction in *VR* [98], virtual interfaces [9], and video game control [99].

Hand pose estimation has also many applications within social and medical sciences, as well as simulation and training, as depicted in Figure 1.4. Hand gestures can be analyzed to understand social interactions [66] or to detect autism in children [116]. There are also plentiful medical applications [281] that require non-touch interfaces due to the risk of infections. Further, hand pose estimation can be used for simulation-based training [259], since simulation-based training can be more efficient and less dangerous than training

certain procedures in the real world environment.

(a) Detecting social signals.

(b) Medical applications.

(c) Simulation-based training.

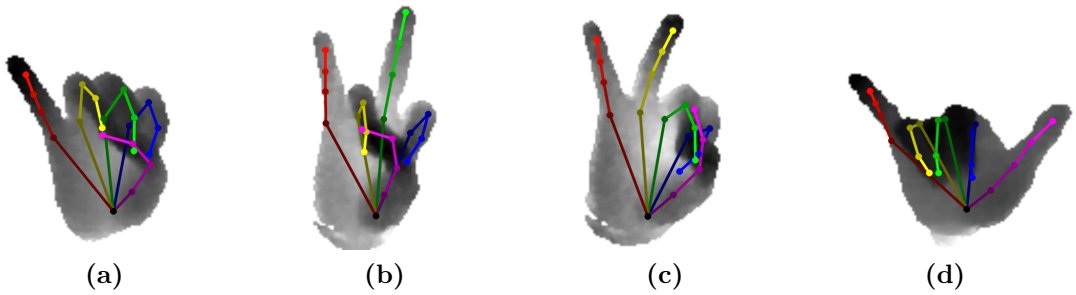
Figure 1.4: Applications of hand pose estimation in medical sciences and training, including analysis of social signals [116], medical hands-free applications [281], and simulation-based training [259].

1.3 Challenges

Markerless 3D hand pose estimation is a hard problem that involves dealing with several challenges. Although some of these challenges are not limited to 3D hand pose estimation and can also be found for example in body pose estimation [229] or facial landmark localization [295], many of these challenges arise in a harder configuration for 3D hand pose estimation.

Many Degrees of Freedom The hand is a highly articulated object as shown in Figure 1.5 and a popular hand model that we introduced in Section 1.1 has 27 *DoF*. Although the hand has 27 *DoF*, they are not independent of each other and there are kinematic constraints that restrict the pose space. Thus, the natural hand articulation does not span the whole 27 *DoF* pose space. Further, the 3D location of the hand wrist is usually factored out by performing localization as a preprocessing step. This removes three *DoF*, but still, there are many remaining *DoF* that need to be estimated. This large pose space is especially difficult for learning-based methods, since they require a large amount of representative training data that ideally covers the full pose space [7].
In addition to the high number of *DoF*, the appearance of the different fingers is very close to each other, *i.e.*, they exhibit self-similarity. This can pose significant problems to methods that only considers small parts of the patch covering the full hand as input, since the different fingers can be easily mixed up.
In case of joint hand-object pose estimation, the object introduces six additional *DoF*, *i.e.*, three *DoF* for 3D rotation and three *DoF* for global 3D translation of the rigid object.

Occlusions Occlusions can pose significant problems to hand pose estimation. While for rigid object the location and pose can be inferred from non-occluded parts of the object,



(a) **(b)** **(c)** **(d)**

Figure 1.5: Challenges of hand pose estimation due to the high number of *DoF*. The hand can be highly articulated as shown in the images, and considering all the possible joint configurations becomes quickly intractable in practice.

this is not possible for the hand due to the complex anatomy with many *DoF* and parts that can move independently. The occluded joint locations have to be inferred from other observations, *e.g.*, by using kinematic constraints or a statistical prior on the pose space. Two types of occlusions can occur in hand pose estimation: (1) occlusions from other objects, such as hand-held objects or other hands, and (2) self-occlusions, where one part of the hand overlaps other parts. Different occlusions are shown in Figure 1.6. Hand pose estimation is especially difficult in egocentric views, where the palm or forearm can completely occlude the fingers. In such a case the exact locations of the occluded joints are undefined when considering the image only. The best thing that we can do is to predict a distribution of possible locations.
These occlusions do not only make inference difficult, but also creating annotations for the training and testing datasets.

(a) Self-occlusion. **(b)** Occlusion. **(c)** Egocentric viewpoint.

Figure 1.6: Challenges of hand pose estimation due to occlusions. Occlusions can be either self-occlusions, or caused by occluders, as for example hand-held objects or other hands. Especially the egocentric viewpoint shows severe self-occlusions.

Size and Shape Human hands show high variability in their size and shape. Not only the size changes for each person significantly between childhood and adulthood, but there are also significant differences between adults. Factors that influence the hand size involve body height, gender, ethnicity, *etc*. Data from an anthropometric survey [78, 162] shows high variations in size. Hand lengths vary from 145 mm to 215 mm for women and 144 mm to 231 mm for men. Also the hand shapes vary significantly, where for example the middle finger PIP circumference ranges between 64 mm and 77 mm for men, and 56 mm to 68 mm for women.

Within the field of computer vision, related works have spent little effort so far on dealing with these variations. This is mostly due to the fact that there is not enough data available for modeling these variations. For example, the dataset of Sun *et al*. [247] contains nine subjects with hand sizes that range from approximately 150 mm to 200 mm, and the dataset of Tompson *et al*. [271] contains two subjects with hand sizes of approximately 200 mm and 250 mm, respectively.

These variations in size and shape lead to different hand appearances as shown in Figure 1.7. This poses challenges to the different pose estimation methods. Generative methods need to address this by adjusting the 3D hand model [239, 252, 266], and discriminative methods, so far, are trained to be invariant to the different shapes [73, 170, 254].

(a) Different hand sizes. **(b)** Different hand shapes.

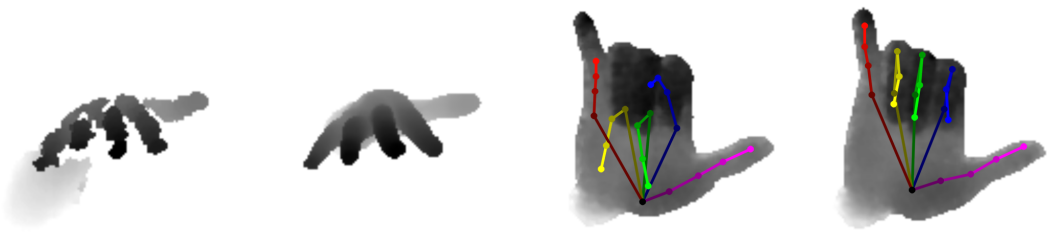
Figure 1.7: Challenges of hand pose estimation due to hand size and shape. The size and shape of the human hand can significantly differ between subjects, which requires special attention.

Noisy Data Methods for 3D hand pose estimation require robustness to noisy data. This noise can be observed at different points within the pipeline.

First, the input to the 3D pose estimation method, *i.e*., the images, can exhibit noise. Since the hand only occupies a small portion of the camera frame, the signal to noise ratio can be low. Further, the images exhibit noise characteristic to the sensor type. For depth image, structured-light sensors exhibits missing regions and noisy outlines [41], and time-of-flight sensors exhibit noise in the depth measurements and hanging pixels along depth discontinuities [106]. In color images there can be shadows and different lighting conditions as well as pixel noise. When comparing real and synthetic data, one can observe

significant discrepancies between them. This domain gap deteriorates the accuracy of the predictions when considering only synthetic data for training and requires special care. Figure 1.8a shows the difference between real and synthetic data.

Second, the dataset annotations can be noisy or erroneous, which can pose difficulties to learning, *i.e.*, the targets for the output of the 3D pose estimation method are noisy. Since large scale dataset annotations are created using automatic annotation procedures [247, 254, 271], there is no guarantee that these annotations are correct. Figure 1.8b shows correct and actual annotations from a popular hand pose estimation dataset [247]. This erroneous annotations can lead to convergence problems during training or cause overfitting to the annotations errors.



(a) Real and synthetic depth images.

(b) Dataset annotation and correct annotation.

Figure 1.8: Challenges of hand pose estimation due to noisy data. There is a large difference in appearance between real and synthetic data as depicted in (a). Also annotations provided along with the dataset can be inaccurate due to automated annotation procedures and (b) shows significant differences between the annotations from the dataset of [247] and the correct annotations.

Fast Motion The hand can move very fast, specifically the wrist can move with up to 5m/s in translation and 300°/s in rotation [61], while the individual fingers can move even faster. For example, closing an open hand at a natural speed takes only 60ms [269]. This fast movement can cause problems at different levels. First, it can cause motion blur or artifacts that deteriorates the quality of the input image, as shown in Figure 1.9. Given video frame rates of common consumer cameras, *i.e.*, around 30 frames per second (fps), this can cause large differences in appearance between two consecutive frames. This is especially challenging for tracking-based methods that use the pose of the previous frame as initialization. Further, these fast movements require methods to have low latency and methods that can run in real-time in order to enable adequate user interactions.

Detection and Segmentation In the hand pose estimation literature, detection and segmentation are often assumed as a prerequisite and provided [73, 170, 254]. While in lab environments simple *ad hoc* approaches such as color or depth thresholding can be used,

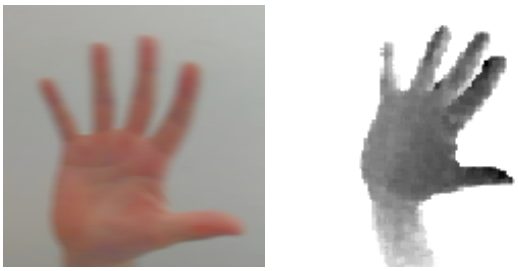


Figure 1.9: Challenges of hand pose estimation due to fast hand motion. Fast motion can cause motion blur in color images and artifacts in depth images, such as thinned structures or distortions of geometric measurements [138]. Ultimately, this can significantly deteriorate the accuracy of the prediction.

in real-world environments this can be a non-trivial task as shown in Figure 1.10. Such depth thresholding techniques fail for example, when the hand gets close to the body, or color thresholding fails when the hand has a similar color to the background. Further, it can be especially difficult to separate the hand from the forearm since they share the same uniform skin color and they are close in depth. Also, background clutter can pose problems to an accurate detection.

Figure 1.10: Challenges of hand pose estimation due to segmentation. The forearm or background clutter might be included in the hand segmentation, which can deteriorate the accuracy of the prediction.

1.4 Strategies

As discussed in the previous sections, there are many practical applications of 3D hand pose estimation and due to the many challenges it is still an unresolved problem raising considerable scientific interest within the last years. Consequently, there are many different approaches towards solving this hard problem.

A first classification of such approaches can be done by distinguishing between invasive methods, which require additional hardware attached to the users hand, and non-invasive methods, which are based on a remote sensing technology, such as radar [147], or cam-

eras [50, 254, 271]. Invasive methods include for example gloves [53], exoskeletons [81], or marker-based tracking environments [88]. While such invasive methods deliver fast and accurate results, their practical applicability is limited due to additional expensive setups and hardware that hinders free hand motion. Many of these artificial input devices could be replaced by non-invasive bare-hand interactions.

Non-invasive methods are most practical for everyday users and camera-based methods are the most popular research direction among them. Computer Vision-based methods can rely on cheap and ubiquitous cameras and do not restrict the users hand movement by additional hardware. Although being most appealing for applications, using non-invasive camera-based methods requires solving the aforementioned hard problems.

(a) Windows Mixed Reality motion controllers.

(b) Oculus Rift Touch.

(c) HTC Vive controllers.

(d) Leap Motion controller.

Figure 1.11: Popular interaction devices from industry, as for example the Windows Mixed Reality motion controllers [166], Oculus Rift Touch [185], HTC Vive controllers [101], and Leap Motion controller [137].

The industry so far relies mostly on invasive input devices and several examples are as shown in Figure 1.11. Popular devices include the Windows Mixed Reality motion controllers [166], Oculus Rift Touch [185] and HTC Vive controllers [101]. The most popular non-invasive device is the Leap Motion controller [137]. Although it works with bare hands, it relies on a complex hardware with stereo camera and stereo illumination that consumes much power and limits its applicability on untethered devices.

In this thesis, we focus on the more challenging Computer Vision-based methods, which can be further separated into discriminative frame-based and generative tracking-based methods. Discriminative methods learn a mapping from an observed image showing the users hand to the hand pose representation and can be applied to a single frames [52, 82, 173, 178, 300, 321, 325]. Generative methods require a hand model and work on image sequences [186, 187, 201, 227]. They require the previous frames' pose as initialization of the current frame [165, 187, 240, 279] and solve an optimization problem such that the articulated hand model matches the observation in the image. However, such methods still require an initialization for the first frame [187, 224], and are prone to tracking failures, especially when the difference in appearance between the current and previous frame is large due to fast hand movement. With the raise of large

scale datasets and efficient learning methods, discriminative methods became tractable, and within this thesis we focus on discriminative methods. Our research showed that we can achieve the same accuracy as generative methods but without requiring a hand model, without solving an optimization problem for each frame at run-time, and more importantly without the risk of tracking failures over time.

1.5 Contributions

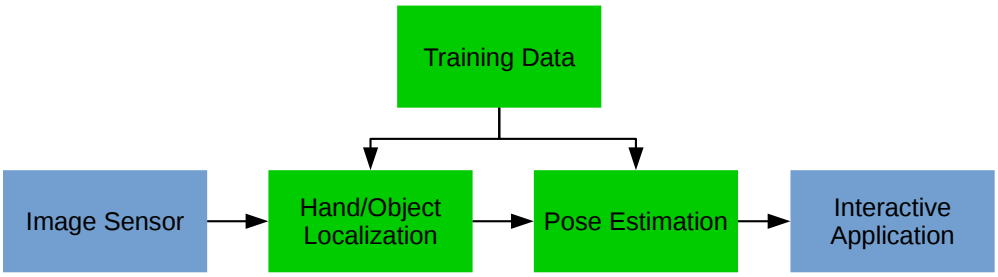


Figure 1.12: General pose estimation pipeline. An image sensor captures data, *e.g.*, color or depth images, that subsequently get processed. The object of interest, *i.e.*, a hand or an object, is localized within the camera frame. Based on the location, a small patch containing only the hand or the object is extracted and used for pose estimation. The estimated pose can then be used for interactive applications, *e.g.*, object manipulation in *VR*. In this thesis, both the localization and pose estimation are based on machine learning methods and thus require training data. This thesis makes contributions in the areas that are highlighted in green.

This thesis contributes in several areas of 3D hand pose estimation and joint hand-object pose estimation. A general pipeline for pose estimation is shown in Figure 1.12 with the areas of contribution highlighted. We divide our contributions in three major parts: First, we present methods for high accuracy and fast 3D hand pose estimation. Second, we show how we can generate training data for learning-based 3D hand pose estimation methods and how we can leverage additional data sources to increase the accuracy or use different modalities as input to our methods. Third, we present a method for joint hand-object pose estimation and show how to robustly estimate the object pose in case of large occlusions that appear frequently when a hand is interacting with an object.

We refer to Section 1.7 for a list of publications and information on where these contributions were originally published. Notably, our 3D hand pose estimation methods presented herein outperformed state-of-the-art methods at the time of publication in terms of accuracy on the popular benchmark dataset [271] twice in 2015 [183] and [184], in 2017 [181], and in 2018 [203]. Also, the original works on which this thesis is based on have already shown great impact in the field of 3D hand pose estimation. Our publications [183] and [184] were cited over 100 times at the time of writing, and [182] and [181] were

cited over 30 times, according to Google Scholar.

1.6 Outline

In the following we describe the overall outline of this thesis, which is structured into three parts.

Part I: Fast and Accurate 3D Hand Pose Estimation In the first part we present methods that can accurately predict the 3D hand pose estimation. Not only these methods are very accurate, they perform 3D hand pose estimation very efficient, by running at hundreds of frames per second. Due to their recent success in Computer Vision, we focus on Deep Learning-based methods, specifically using Convolutional Neural Networks (CNNs) for 3D hand pose estimation. We contribute to these methods twofold.
In Chapter 4 we show that we can efficiently integrate a prior on the 3D pose into a Deep Learning-based pose estimator. This significantly increases the accuracy of the predictions and restricts the predictions to the manifold of feasible 3D hand poses, which is an appealing property in practice.
Further, we show in Chapter 5 that we can learn another predictor that can correct the mistakes made by an initial pose estimator. This predictor takes the original image as input together with a synthesized image of the hand. This additional input introduces feedback by following an analysis-by-synthesis approach [175]. The predictor then predicts an update for the hand pose such that the synthesized image matches better the original input image. This predictor can be applied in an iterative way where each iteration increases the accuracy of the prediction.

Part II: Training Data for 3D Hand Pose Estimation Training the methods presented in Part I requires a large number of training samples, which can be hard to acquire. Therefore, we present a semi-automatic approach for easily creating 3D training data in Chapter 6. A user is asked to perform hand gestures in front of and camera and an image sequence is recorded. From this sequence, we select key frames, *i.e.*, a subset of all frames that contain distinctive hand poses. These key frames are then manually annotated in 2D, which is a much simpler task than annotating them in 3D. We show that users can do this efficiently and with high accuracy. Finally, the 3D hand poses for the key frames are automatically propagated to the remaining frames, which gives us a large number of frames with 3D annotations.
We further investigate in Chapter 7 how additional synthetic training data can be used to increase the accuracy of the 3D pose estimation. Synthetic data is much simpler to acquire than real data, however, there exists a domain gap between the different data sources. This domain gap deteriorates the accuracy of the prediction when using only synthetic data for training the pose estimator. We, therefore, introduce a novel method that transfers features from the real domain to features from the synthetic domain by

using pairs of synthetic and real images. We integrate this approach into our method presented in Chapter 4 and show that this additional data source significantly increases the accuracy of the estimator.
Finally, we show in Chapter 8 how we can transfer the annotations from synthetic depth to real color images. This is a desirable property, since creating synthetic depth images is rather simple: no texture and lighting has to be considered. Further, estimating the 3D hand pose from color images is more practical since it does not require additional depth sensors. Therefore, we introduce a method that uses pairs of unlabeled color and depth images and maps the color features to the features of the synthetic depth images. We show that our method—trained without color labels—achieves similar accuracy compared to methods that are trained with color labels.

Part III: Hands and Objects In the last part of this thesis we investigate hands interacting with objects, which is a realistic scenario that frequently occurs in *AR* and *VR* applications. We, therefore, extend the feedback method presented in Chapter 5 and generalize it to multiple objects in Chapter 9. We extend our method to jointly estimate the hand pose and the object pose. The method comprises two predictors, one for the hand pose and one for the object pose. In a first step the two poses are estimated separately, and then jointly refined in an iterative manner. Feedback is generated after each iteration for the hand and the object by synthesizing images of the hand and the object, respectively. The synthesized feedback of the hand and the object are combined before feeding it to the next iteration. For synthesizing the images, models for the hand and the object are acquired beforehand.
However, objects are often severely occluded by hands or other objects, and dealing with such occlusions requires additional knowledge of the occluder, for example in form of object models, and/or knowledge of the occluded parts. Especially joint hand-object pose estimation becomes increasingly complex with a larger number of interacting objects, *i.e.*, occluders. Pose estimation methods that do not require prior knowledge of the occluders are beneficial in practice since we cannot know all possible occluders beforehand. Therefore, we present a novel approach in Chapter 10 that does not require any prior information about the occluder. Our approach estimates the pose of the object from partial observations that are not corrupted by an occlusion, and robustly combines the individual estimates to compute the final pose.

Before we present our contributions in detail, we first discuss related work, and introduce the datasets and evaluation metrics that are used for evaluating our methods.

1.7 List of Publications

My work at the Institute for Computer Graphics and Vision led to the following peer-reviewed publications. For the sake of completeness of this thesis, they are listed in chronological order along with the respective abstracts.

Hands Deep in Deep Learning for Hand Pose Estimation

Markus Oberweger, Paul Wohlhart, and Vincent Lepetit
In: *Proceedings of Computer Vision Winter Workshop (CVWW)*
February 2015, Seggau, Austria
(Accepted for oral presentation)

Abstract: We introduce and evaluate several architectures for Convolutional Neural Networks to predict the 3D joint locations of a hand given a depth map. We first show that a prior on the 3D pose can be easily introduced and significantly improves the accuracy and reliability of the predictions. We also show how to use context efficiently to deal with ambiguities between fingers. These two contributions allow us to significantly outperform the state-of-the-art on several challenging benchmarks, both in terms of accuracy and computation times.

Training a Feedback Loop for Hand Pose Estimation

Markus Oberweger, Paul Wohlhart, and Vincent Lepetit
In: *Proceedings of International Conference on Computer Vision (ICCV)*
December 2015, Santiago, Chile
(Accepted for oral presentation)

Abstract: We propose an entirely data-driven approach to estimating the 3D pose of a hand given a depth image. We show that we can correct the mistakes made by a Convolutional Neural Network trained to predict an estimate of the 3D pose by using a feedback loop. The components of this feedback loop are also Deep Networks, optimized using training data. They remove the need for fitting a 3D model to the input data, which requires both a carefully designed fitting function and algorithm. We show that our approach outperforms state-of-the-art methods, and is efficient as our implementation runs at over 400 fps on a single GPU.

Efficiently Creating 3D Training Data for Fine Hand Pose Estimation

Markus Oberweger, Gernot Riegler, Paul Wohlhart, and Vincent Lepetit
In: *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*
June 2016, Las Vegas, USA
(Accepted for poster presentation)

Abstract: While many recent hand pose estimation methods critically rely on a training set of labelled frames, the creation of such a dataset is a challenging task that has been overlooked so far. As a result, existing datasets are limited to a few sequences and individuals, with limited accuracy, and this prevents these methods from delivering their

full potential. We propose a semi-automated method for efficiently and accurately labeling each frame of a hand depth video with the corresponding 3D locations of the joints: The user is asked to provide only an estimate of the 2D reprojections of the visible joints in some reference frames, which are automatically selected to minimize the labeling work by efficiently optimizing a sub-modular loss function. We then exploit spatial, temporal, and appearance constraints to retrieve the full 3D poses of the hand over the complete sequence. We show that this data can be used to train a recent state-of-the-art hand pose estimation method, leading to increased accuracy.

DeepPrior++: Improving Fast and Accurate 3D Hand Pose Estimation

Markus Oberweger and Vincent Lepetit
In: *Proceedings of International Conference on Computer Vision Workshops (ICCVW)*
October 2017, Venice, Italy
(Accepted for poster presentation)

Abstract: DeepPrior is a simple approach based on Deep Learning that predicts the joint 3D locations of a hand given a depth map. Since its publication early 2015, it has been outperformed by several impressive works. Here we show that with simple improvements: adding ResNet layers, data augmentation, and better initial hand localization, we achieve better or similar performance than more sophisticated recent methods on the three main benchmarks (NYU, ICVL, MSRA) while keeping the simplicity of the original method. Our new implementation is publicly available.

Feature Mapping for Learning Fast and Accurate 3D Pose Inference from Synthetic Images

Mahdi Rad, Markus Oberweger, and Vincent Lepetit
In: *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*
June 2018, Salt Lake City, USA
(Accepted for poster presentation)

Abstract: We propose a simple and efficient method for exploiting synthetic images when training a Deep Network to predict a 3D pose from an image. The ability of using synthetic images for training a Deep Network is extremely valuable as it is easy to create a virtually infinite training set made of such images, while capturing and annotating real images can be very cumbersome. However, synthetic images do not resemble real images exactly, and using them for training can result in suboptimal performance. It was recently shown that for exemplar-based approaches, it is possible to learn a mapping from the exemplar representations of real images to the exemplar representations of synthetic images. In this paper, we show that this approach is more general, and that a network can also be applied after the mapping to infer a 3D pose: At run-time, given a real

image of the target object, we first compute the features for the image, map them to the feature space of synthetic images, and finally use the resulting features as input to another network which predicts the 3D pose. Since this network can be trained very effectively by using synthetic images, it performs very well in practice, and inference is faster and more accurate than with an exemplar-based approach. We demonstrate our approach on the LINEMOD dataset for 3D object pose estimation from color images, and the NYU dataset for 3D hand pose estimation from depth maps. We show that it allows us to outperform the state-of-the-art on both datasets.

Making Deep Heatmaps Robust to Partial Occlusions for 3D Object Pose Estimation

Markus Oberweger, Mahdi Rad, and Vincent Lepetit
In: *Proceedings of European Conference on Computer Vision (ECCV)*
September 2018, Munich, Germany
(Accepted for poster presentation)

Abstract: We introduce a novel method for robust and accurate 3D object pose estimation from a single color image under large occlusions. Following recent approaches, we first predict the 2D projections of 3D points related to the target object and then compute the 3D pose from these correspondences using a geometric method. Unfortunately, as the results of our experiments show, predicting these 2D projections using a regular CNN or a Convolutional Pose Machine is highly sensitive to partial occlusions, even when these methods are trained with partially occluded examples. Our solution is to predict heatmaps from multiple small patches independently and to accumulate the results to obtain accurate and robust predictions. Training subsequently becomes challenging because patches with similar appearances but different positions on the object correspond to different heatmaps. However, we provide a simple yet effective solution to deal with such ambiguities. We show that our approach outperforms existing methods on two challenging datasets: The Occluded LineMOD dataset and the YCB-Video dataset, both exhibiting cluttered scenes with highly occluded objects.

Domain Transfer for 3D Pose Estimation from Color Images without Manual Annotations

Mahdi Rad, Markus Oberweger, and Vincent Lepetit
In: *Proceedings of Asian Conference on Computer Vision (ACCV)*
December 2018, Perth, Australia
(Accepted for oral presentation)

Abstract: We introduce a novel learning method for 3D pose estimation from color images. While acquiring annotations for color images is a difficult task, our approach

circumvents this problem by learning a mapping from paired color and depth images captured with an RGB-D camera. We jointly learn the pose from synthetic depth images that are easy to generate, and learn to align these synthetic depth images with the real depth images. We show our approach for the task of 3D hand pose estimation and 3D object pose estimation, both from color images only. Our method achieves performances comparable to state-of-the-art methods on popular benchmark datasets, without requiring any annotations for the color images.

Generalized Feedback Loop for Joint Hand Object Pose Estimation

Markus Oberweger, Paul Wohlhart, and Vincent Lepetit
In: *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*
(Currently under review)

Abstract: We propose an approach to estimating the 3D pose of a hand, possibly handling an object, given a depth image. We show that we can correct the mistakes made by a Convolutional Neural Network trained to predict an estimate of the 3D pose by using a feedback loop. The components of this feedback loop are also Deep Networks, optimized using training data. This approach can be generalized to multiple objects, and we demonstrate this specifically for a hand interacting with an object. In that case, we jointly estimate the 3D pose of the hand and the 3D pose of the object. Our approach performs en-par with state-of-the-art methods for 3D hand pose estimation, and outperforms state-of-the-art methods for joint hand object pose estimation when using depth images only. Also, our approach is efficient as our implementation runs in real-time on a single GPU.

2 Related Work

Contents

This thesis aims for Computer Vision-based 3D hand pose estimation and joint hand-object pose estimation. In this chapter we review related literature and put our contributions in context to existing approaches. In a broader context, 3D hand pose estimation is related to other structured keypoint estimation methods, such as facial landmark localization and body pose estimation. We give a short review of important and related works in these areas, before we give a more detailed review of related works in the field of hand pose estimation. Finally, we review works on joint hand-object pose estimation.

2.1 Structured Keypoint Localization

Hand pose estimation is related to other keypoint localization tasks, such as facial landmark localization or body pose estimation. We therefore give a short overview of related works in these fields.

Facial Landmark Localization Facial landmark localization aims at localizing distinctive facial features, such as the corners of the eyes, nose tip, pupil center, *etc.* An example of annotated landmarks is shown in Figure 2.1a. Facial landmark localization is a prerequisite for many applications, *e.g.*, expression analysis [5], face animation and reenactment [70], or face registration [174]. Although the key challenges of facial landmark estimation are similar to hand pose estimation, facial landmark localization is simpler: The face has less Degrees of Freedom (DoF) [10] than the hand and thus less self-occlusions.

The face is not articulated and therefore no kinematic rules are required. Since the face is approximately symmetric, the occluded parts can be reconstructed from the visible parts [294]. Also, the landmark locations are more structured and only locally deformable. Fitting an average face and performing local adjustments is sufficient for accurate landmark localization [18]. Due to the large corpus of works in facial landmark localization, we only consider more related work and refer to [235, 287] for a broader overview of the field.

In the seminal work of Cootes *et al.* [44], the Active Appearance Model (AAM) was introduced for fitting a deformable object model to an image. The AAM is a statistical model that is learned from a collection of annotated images, such as faces with landmarks, and models the shape and appearance of a specific object category. This generative model is parametrized by a set of orthogonal variables. In order to fit the AAM to an image, the variables are iteratively optimized by minimizing the difference in appearance between the observed image and the rendered model. However, due to limited expressive power of the model and its holistic representation, it does not generalize well to unseen objects. Some of these weaknesses were addressed in later work, such as using a non-linear shape model [59], or more efficient iterative optimization [221, 273].

Recent methods follow a cascaded or iterative approach that rely on local representations. The Supervised Descent Method (SDM) [298] represents one of the seminal works of this class of approaches. They initialize the landmarks with the average locations and extract local features, specifically SIFT features [155], around these locations. Based on these local features, they formulate an iterative refinement as a non-linear least squares problem, which can be solved with the Newton method for example, that predicts gradient directions for all landmarks jointly. The landmark locations are iteratively updated based on the predicted gradients. Following such an iterative regression, [210] learns a set of binary features that encode the local appearance of the landmark. They combine the local features in a global regressor that predicts an update on the landmark locations, which can be run very efficiently. [33, 119] use a cascade of regression trees that are trained to predict updates of the landmark locations in each step of the cascade. The trees not only learn the updates, but also localized image features depending on the current landmark locations.

More recent works rely on the large learning capacity of Convolutional Neural Networks (CNNs). [324] uses an iterative approach where a cascade of *CNNs* predict updates on the parameters of a face model. The *CNN* takes as input a color coded rendering of a 3D face model together with the input image. The network learns to predict an update for the parameters of the model such that the rendered image aligns well with the input image. [135] uses a single *CNN* to extract features from the image and an additional simple network to predict initial landmark locations. Features are extracted around the landmark locations and fed to a Long Short-Term Memory (LSTM) network that iteratively predicts updates on the landmark locations.

(a) Facial landmark localization.

(b) Body pose estimation.

Figure 2.1: Structured keypoint localization. Hand pose estimation shares similar challenges with, *e.g.*, facial landmark localization [318] shown in (a) with landmarks visualized as white dots, or body pose estimation [290] shown in (b) with skeleton shown as colored sticks.

Body Pose Estimation Body pose estimation aims at estimating the configuration of body parts and is also related to hand pose estimation. Many hand pose estimation approaches are rooted in the field of body pose estimation. Thus, we shortly review this field here as well. An example image with estimated human joint locations is shown in Figure 2.1b. Body pose estimation shares many challenges with hand pose estimation such as a large number of *DoF* with kinematic constraints, and large variations in size and shape. However, the human body does not show self-similarity except the symmetry and in general less self-occlusions. Also the movement is much slower compared to the hand. Human pose estimation has also unique challenges such as a larger variation in appearance due to clothing, which does not apply to the hand that is usually bare. Again, we only report the most relevant works in this section and refer to [37, 168, 199] for a more in-depth review.

For body pose estimation there are already commercial products available for a few years now, such as the method integrated in the Microsoft Kinect. This technology relies on the seminal work of Shotton *et al.* [229], who presented an approach that predicts the 3D joint locations from depth images. They use a per-pixel semantic labeling of the body parts, which is predicted using a Random Forest with depth-dependent features. They further refine the joint locations by regressing offsets for the joints. Their approach is enabled by a large scale synthetic dataset used for training. [260] infer dense correspondences between a depth image and a 3D model by predicting coordinates corresponding to the surface location of the model. From these predicted correspondences they perform optimization to obtain the pose.

Many popular approaches for monocular human pose estimation build upon the Pictorial Structures (PS) model [64]. [206] learns a deformable part-based model and use a probabilistic inference to match the body parts to edges extracted from the image. They iteratively parse the image by assigning soft labels to the image regions. In each iteration they learn a foreground/background color model to refine the part detection. Finally, they use the highest likelihood locations as the joint detections. [65] extended the *PS* model to image sequences by introducing additional spatio-temporal constraints. They also use foreground/background segmentation to restrict the spatial search space for a more efficient inference. [6] combines strong discriminative part detectors with a prior on the kinematic tree and models the part dependencies as Gaussian distributions for efficient inference. [48] extended the *PS* model with a two-step procedure. In the first step they learn discriminative, independent body part classifiers. In the second step they apply a regressor to predict joint positions depending on the initial body part distributions. The regressor learns the dependencies between the different joints to solve for ambiguities. [115] replaced the part detectors of the *PS* model by a cascade of part detectors, each applied to more likely part regions of the previous stage. Thus they can apply the detectors in a step by step refinement. [13] uses multiple views to perform 3D pose estimation. They apply the *PS* method in 2D for each view and combine the different views for 3D inference. They use a discrete state-space for efficiency. [231] presented a solution to handle occlusions in the *PS* framework, by explicitly encoding the occlusions between the body parts at the pixel level. [304] introduced a novel representation of the part model that incorporates contextual co-occurrence between parts using a tree-structured spring model. [23] introduced poselets, which represent parts that are tightly clustered in appearance and pose space. Poselets are detected in the image and combined to infer the part locations and pose. Since pose estimation is multi-modal, *i.e.*, the part configurations are in different spatial modes, [220] uses different models for each mode. The modes are filtered via a cascaded prediction step and then local mode-specific model are applied.

With the advent of more powerful models, such as *CNNs*, it became feasible to directly predict the human pose from monocular color images. [272] directly predicts the 2D joint locations using a regression *CNN* and further refines the joint locations by learning a cascade of joint regressors that predict updates for the joint locations. [270] jointly trains a *CNN* and graphical model. The *CNN* is used as a part detector and the graphical model incorporates the spatial dependencies between the different parts, which, however, is not effective for high variations in pose space. A similar spatial model to enforce global pose consistency was used in [113]. Recently, direct prediction of the 3D body pose from monocular images became feasible by addressing the inherent ambiguities with large-scale training data. [107] combines body part labeling and iterative structured-output regression for 3D joint localization. Also, powerful *CNN* models can be used to directly learn 3D pose estimation from monocular color images [163, 196, 263], and even from challenging egocentric viewpoints [301].

Another line of work relies on a model of the human body. [71] uses a simple 3D stick

figure as a body model. They render the outline of the model and maximize the similarity of edge contours between the rendered and observed image using the Chamfer distance. For efficient optimization they decompose the search space and perform independent optimization of torso and limbs. [110] uses a simple subdivision model for the body and applies Particle Swarm Optimization (PSO) to optimize multi-view silhouettes to obtain the human pose. *PSO* is also used for the optimization in [326], who use a 2D statistical shape model of the body. They infer the pose for image sequences by keeping track of the parts using optical flow between consecutive frames, and constraint the part locations with an additional hand detector. They optimize the pose likelihood over the full sequence and take the pose with highest probability per frame. [20] first estimates the 2D body joint locations using a *CNN*-based method and then uses a statistical body shape model fit to these prediction. In the optimization they penalize differences in the projected 3D model joints to the detected 2D joints. Using the 3D model allows to account for interpenetrations and physical feasibility. [164] uses a fully-convolutional network to jointly regress the 2D and 3D joint locations, followed by a kinematic skeleton fitting method for temporally stable pose estimates. Recently, [118] proposed an end-to-end solution that recovers pose and shape of the body simultaneously, by directly predicting the pose and shape parameters of a 3D model. In order to obtain realistic shape parameters, they jointly train a discriminator that distinguishes between real and unreal human shapes. By using a 3D joint reprojection loss, they only require 2D annotations for color images that are easier to acquire.

2.2 Hand Pose Estimation

Hand pose estimation shares several of the challenges with other structured keypoint estimation problems, as discussed in the previous section, but nevertheless has its own large corpus of distinctive work. Hand pose estimation is an old problem in Computer Vision, with early references from the nineties [108, 209], but it is currently very active probably because of the appearance of depth sensors and its application in Augmented Reality (AR) and Virtual Reality (VR). There is a significant amount of early work that deals with hand pose estimation and we refer to Erol *et al.* [61] for an overview.

Here we discuss only more recent work, which can be divided into two main types of approaches: Generative and discriminative methods. Figure 2.2 illustrates the fundamental differences. While both methods aim at assigning a pose in an output space to an observation in an input space, the way this assignment is performed is fundamentally different. In this work, the input space corresponds to depth or color images and the output space corresponds to the 3D poses.

Generative methods require an initialization for the pose, *e.g.*, the pose of the previous frame [251, 259], and perform an optimization that minimizes the difference between the observed data and the model. This can be achieved for example by rendering a model under the current pose and comparing this rendering with the input image [187, 299]. We

present more details on this type of approaches in Section 2.2.1.
Discriminative methods rely on a mapping between the input space and the output space. There are various ways of implementing this mapping or even learning it from training data as we discuss in more detail in Section 2.2.2.

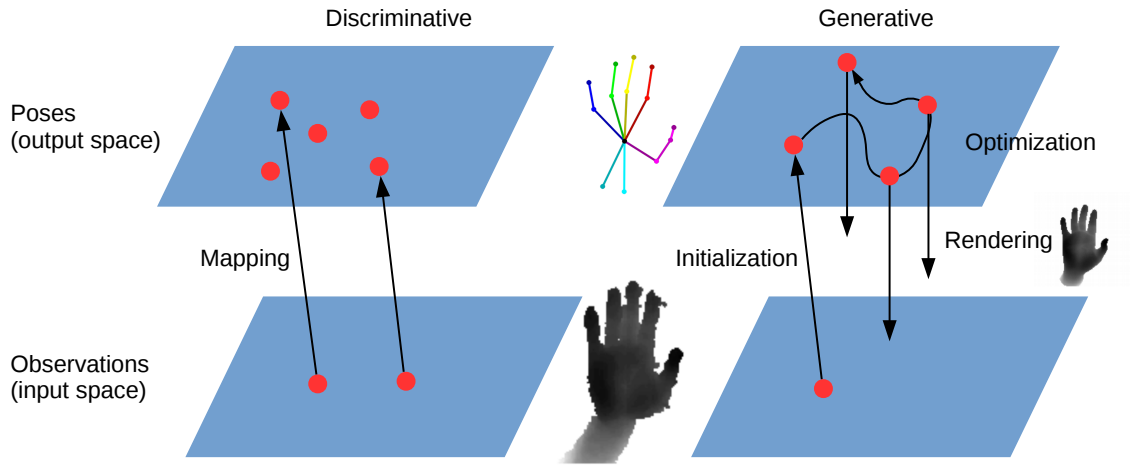
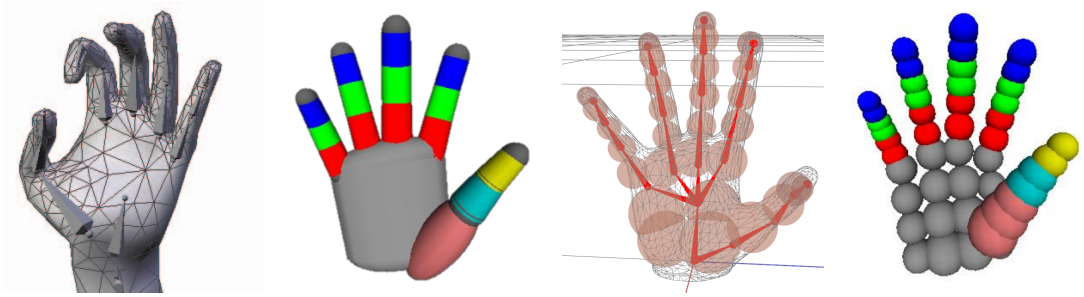


Figure 2.2: Discriminative and generative methods. On the one side we have the observations, *i.e.*, the input space or input images, that we want to assign a pose in the output space. While discriminative methods learn a mapping from the input images to the poses, generative methods perform an optimization of the pose such that an articulated model resembles the input data as accurate as possible.

2.2.1 Generative Methods

The first type of approaches are based on generative, model-based tracking methods. The works of this type are developed from four generic building blocks: (1) a hand model, (2) a similarity function that measures the fit of the observed image to the model, (3) an optimization algorithm that maximizes the similarity function with respect to the model parameters, and (4) an initial pose from which the optimization starts.

Hand Model There is a large variability of hand models used in the literature. The choice of a simple model is important for maintaining real-time capabilities, and representing a trade-off between speed and accuracy as a response to a potentially high degree of model abstraction. Depending on the used optimization algorithm, additional requirements for the model might be imposed, such as differentiability [241], local rigidity [265] or mesh interpolation [259].
A large number of different hand models was proposed that use different hand-crafted geometrical approximations for the hand and some popular models are depicted in Figure 2.3. [187, 189, 224, 251] use geometric primitives, such as cylinders, ellipsoids, spheres, and cones to model the hand. [128] uses simple prism shapes, and [201] uses a model con-



(a) 3D mesh [50]. **(b)** Geometric primitives [187]. **(c)** Sum of Gaussians [242]. **(d)** Spheres [201].

Figure 2.3: Generative hand models. The different hand models show different levels of abstraction, ranging from a fully rigged 3D mesh [50], an assembly of simple geometric primitives [187], a Sum of Gaussians [242], to multiple spheres [201].

sisting of spheres. Similarly, [265] also uses spheres, but the spheres are combined with the union convolution primitive operator to obtain a differentiable shape with a smooth surface. [241] models the hand from a Sum of Gaussians, which is also differentiable. More holistic hand representations are used by [12, 145, 225, 227, 255, 279] utilizing a mesh-based Linear Blend Skinning (LBS) model of the hand. [259] uses a 3D mesh with a subdivision surface that allows efficient interpolation of surface vertices. [299] increases the matching quality by using depth images that resemble the same noise pattern as the depth sensor. While the previous works deal with depth images or silhouettes that are simple to render, [50] uses a fully shaded and textured triangle mesh controlled by a skeleton.

In order to work well in practice, these hand models should be adjusted to the users hand [239, 252, 266], which requires an additional optimization step or manual adjustment of hand shape parameters [165]. [239] adjusts the hand length, width, and finger thickness by modifying the position and variance of their Sum of Gaussians hand model. They perform a greedy search over a fixed range of parameters and select the parameters that best explain the image evidence. [261] adapts a rough template hand model to different users by adjusting the vertices of the template by fitting it to a small number of depth frames captured from the users hand. [124] extended this work and represents the hand shape parameters as a linear combination of a mean mesh with a number of offset vectors. Again, these shape parameters are optimized over a short sequence of frames to adjust the model to a user [252]. Therefore, they minimize the difference between a rendered depth image from the hand model and the observed depth image. They use a gradient-based optimizer together with a geometric prior, but they require a rough pose initialization for each frame in the sequence. [266] performs the model adjustment in parallel with the hand tracking. They adjust the radii and positions of their sphere-based hand model by running a gradient-based optimization.

Similarity Function Different modalities were proposed for the similarity function, which should measure the fit of the hand model to the input data and thus the similarity function is coupled to the used model. Depth-based methods naturally rely on similarity functions that consider depth values or point clouds, and apply the ℓ_1 or ℓ_2 norm, or truncated versions thereof [128, 165, 186, 201, 225, 255, 299]. [279] uses salient points in addition to the depth, [189, 241] combine color and depth, and [251] combines the point cloud with the hand silhouette for rough alignment. [50] evaluates the difference between color images considering shading and texture, and [187] uses edges and color. [12] relies on salient points, edges, and optical flow. [224] relies on the signed distance function for efficiently finding correspondences between input depth and model points, and [145] uses a semantic labeling of the different fingers of the hand together with the point cloud. Within this context of plentiful choices of distance measurements, [277] presented an evaluation of the performance of different distances.

Optimization The optimization of the similarity function is a critical part, as the high-dimensional pose space is prone to local minima. Thus, *PSO* is often used to handle the high dimensionality of the pose vector [186, 187, 201, 227]. *PSO* maintains a set of hypothesis that are sampled across the search space. Thereby, each hypothesis represents a full hand pose. The similarity function is evaluated for each hypothesis and the hypothesis is updated according to its own best solution and the best solution among the full set of hypothesis. Although this is an appealing algorithm that can be easily parallelized and thus efficiently implemented, it requires a good initialization and does not guarantee to find the global optimum [317].
[189] proposed an evolutionary search algorithm similar to *PSO*. They start from a densely sampled search space and identify the best fitting hypothesis. They iteratively reduce the size of the search space and increase the space resolution, and each iteration starts from a weighted average of the previous best hypothesis.
Due to the high computation time of these optimization methods, which have to be solved for every frame, [299] does not optimize the pose but only evaluates the similarity function for several hypothesis to select the best hypothesis. [255] proceeds along the skeleton tree and use a Random Forest to predict different hypothesis of the child joints within the skeleton tree. This allows to sample hypothesis of the different fingers and does not require a holistic pose hypothesis.

Differently, [12, 50, 224, 241] use gradient-based methods to optimize the pose, while [165] uses dynamics simulation of the 3D model. Most notable, [50] deals with color images and thus cannot resort to geometry methods to optimize the similarity function. They require a carefully designed hand model that supports differentiation of the model with respect to the pose. They calculate gradients to update illumination, texture and pose using a local optimization method.

For optimizing a similarity function that is based on point clouds, geometric methods such as Iterative Closest Point (ICP) are often employed. [251] uses an articulated *ICP*

formulation that is optimized using the Levenberg-Marquardt algorithm and constraint with a prior on the hand pose, together with kinematic and temporal constraints. [145] optimizes *ICP* that is constrained with an inverse kinematic pose obtained from detected finger tips. [128] uses a formulation similar to *ICP*, which employs point to surface constraints that are predicted by a Random Forest. Optimizing these constraints pulls the model into position, thus minimizing the error between model and observation. [225] applies inverse kinematics from correspondences of target points that are matched to their spatially closest points on the surface of the hand model. They optimize the distance between the corresponding points by applying a linearization to the Gauss-Newton approach with additional bounds on the joint angles and a prior on the hand pose.

Recent works aim at handcrafting a differentiable similarity function, which tightly integrates the hand model to achieve accurate and fast results [259, 265]. Such an formulation allows for directly calculating the gradients of the pose with respect to the model, however, requires careful design and implementation.

Pose Initialization In order to kick-start optimization and to guarantee the convergence to a meaningful pose, a strong prior on the initial pose of the optimization is required. The simplest choice is a predefined initial hand location and pose [50], or a manual initialization of the first frame [187, 224], although this not really useful in practice. More useful are automatic methods that provide an initial pose. [227, 241] use a discriminative part-based pose initialization, and [201] uses a finger tip detector. [299] predicts candidate poses using a Hough Forest. Furthermore, tracking-based methods use the pose of the previous frame to initialize the consecutive frame [165, 187, 240, 279]. This can be problematic if the difference between the frames is too large, *e.g.*, due to fast motion or low frame rates.

2.2.2 Discriminative Methods

The second type of approaches is based on discriminative models that aim at directly predicting the joint locations from color, depth, or RGB-D images. The accuracy and speed of such methods critically rely on the mapping from image to pose.

Early works formulated hand pose estimation as a database retrieval problem. Therefore, a database of synthetic images is generated in order to identify the samples that are most similar to a query image. [289] uses a color glove with a specific pattern and perform a nearest-neighbor search by comparing tiny color images. [228] matches the silhouette with the database, and [217] uses silhouette, color segmentation and a simple feature mapping learned from shape moments. [8] uses a probabilistic edge matching to retrieve relevant database samples, and [54] matches the depth and edges for retrieval. A disadvantage of such approaches is that the pose accuracy is limited to the number of samples in the dataset. For hand pose estimation with many DoF, the number of required samples can quickly become intractable [7] and prohibitively slow. To overcome the effi-

ciency problem, [249] proposed a fast nearest-neighbor method that relies on a volumetric Hamming distance to jointly detect the hand in a sliding window fashion and estimate the pose. [40] identifies the nearest neighbors from a database of hand poses using local shape descriptors. The poses of the nearest neighbors are then used to retrieve the unknown pose of the hand using joint matrix factorization and completion. [214] formulates hand pose estimation as a multi-class classification problem and uses a coarse-to-fine tree-based approach for classification. Although it works in egocentric environments, they require strong priors over viewpoint and pose, and photorealistic renderings of the egocentric scenes.

Some recent works, including [122, 123, 134, 254, 256], use different approaches with Random Forests, but are restricted to static gestures, showing difficulties with occluded joints, or causing high inaccuracies at finger tips. [123, 134] rely on multi-layered Random Forests for the prediction. [123] uses invariant depth features, and [134] uses clustering in hand configuration space and pixel-wise labeling. However, both do not predict the actual 3D pose but only classify given poses based on a dictionary. Motivated by work for human pose estimation [229], [122] uses Random Forests to perform a per-pixel classification of depth images and then a local mode-finding algorithm to estimate the 2D joint locations. However, this approach cannot directly infer the locations of hidden joints, which are much more frequent for hands than for the human body. [256] proposed a semi-supervised regression forest, which first classifies the hands viewpoint, then the individual joints, to finally predict the 3D joint locations. However, it relies on a costly pixel-wise classification, and requires a huge training database due to viewpoint quantization. The same authors proposed a regression forest in [254] to directly regress the 3D locations of the joints, using a hierarchical model of the hand. However, their hierarchical approach accumulates errors, causing larger errors for the finger tips.

These problems have been addressed by more recent works that resort to *CNNs* to learn the mapping from image to pose [52, 82, 173, 178, 271, 300, 321, 325]. One of the first works that used a *CNN* is [271], who use a *CNN* for feature extraction and instead of predicting the 3D joint locations directly, generate small 2D heatmaps for the different joints. They use the heatmaps to infer the hand pose using inverse kinematics. However, their approach predicts only the 2D locations of the joints, and uses a depth image for the third coordinate, which is problematic for hidden joints. Furthermore, the accuracy is restricted to the heatmap resolution. [72] extended this work and uses multiple *CNNs* to predict heatmaps from different reprojections of the depth image, which requires a separate *CNN* for each reprojection. Also, this approach requires complex post-processing to fit a kinematic model to the heatmaps. Similarly, Wan *et al.* [284] use 2D heatmaps, but in addition they predict heatmaps that encode the proximity of the input 3D points to the joint locations and dense 3D vector offsets pointing towards the 3D joint location. This representation requires complex post-processing to obtain the 3D locations of the joints.

Many recent approaches exploit the hierarchy of the hand kinematic tree. [300] predicts updates along the skeleton tree that correct an initial pose and use Lie-algebra to constrain

these updates. Sun *et al.* [247] estimate the joint locations in normalized coordinate frames for each finger, and [233] uses a separate regressor for each finger to predict spatial and temporal features that are combined in a nearest-neighbor formulation. [307] introduces a spatial attention mechanism that specializes on each joint and an additional optimization step to enforce kinematic constraints. [142] splits the hand into smaller sub-regions along the kinematic tree. [303] predicts a gesture class for each pose and trains a separate pose regressor for each class. [114] relies on the hierarchical topology of the hand and uses a Random Forest for each finger to estimate 3D coordinates of the joints. All these approaches require multiple predictors, one for each joint or finger, and often additional regressors for different iterations of the algorithms. Thus the number of regression models ranges from tens to more than 50 different models that have to be trained and evaluated. This can quickly become problematic with an increasing number of joints to predict.

To overcome this shortcoming, there are several works that integrate the kinematic hierarchy into a single *CNN* structure. Guo *et al.* [82] and similarly Chen *et al.* [39] train an ensemble of sub-networks for different spatial regions of input features, and Madadi *et al.* [156] use a tree-shaped *CNN* architecture that predicts different parts of the kinematic tree. However, this requires a specifically designed *CNN* architecture depending on the used hand annotation. Zhou *et al.* [321] integrate a hand model into a *CNN* by introducing an additional layer that enforces the physical constraints of a 3D hand model. However, these constraints have to be manually defined beforehand.

Different data representations of the input depth image were also proposed. Deng *et al.* [52] convert the depth image to a 3D volume and use a 3D *CNN* to predict joint locations. Similarly, Moon *et al.* [170] represent the hand depth image as a voxel grid and use 3D *CNN* to predict volumetric heatmaps. Also, [73] uses a 3D *CNN* with different volumetric representations to directly predict the 3D joint locations with an additional prior on the 3D hand pose, similar to the one proposed in this work. However, 3D networks show a low computational efficiency [213]. Differently, [285] uses surface normals instead of the depth image, but surface normals are not readily accessible from current depth sensors and thus introduce an additional computational overhead. Neverova *et al.* [178] combine a segmentation of the hand parts with a regression of joint locations, but the segmentation is sensitive to the sensor noise.

A probabilistic framework was proposed by Bouchacourt *et al.* [22], who use a network to learn the posterior distribution of hand poses and one can sample from this distribution. However, it is unclear how to combine these samples in practice. Wan *et al.* [283] use two generative networks, one for the hand pose and one for the depth image, and learn a shared mapping between these two networks, which involves training several networks in a complex procedure.

Fourure *et al.* [67] exploit annotations from different datasets by introducing a shared representation, which is an interesting idea for harvesting more training samples, but has shortcomings when dealing with sensor characteristics. Zhang *et al.* [315] formulate pose estimation as a multivariate regression problem that, however, requires solving a

complex optimization problem during run-time. [38] proposed several different networks for integrated hand detection and pose estimation. [202] extended a single frame pose estimation method to the temporal domain by using the pose from the previous frame for estimating the pose of the consecutive frame.

Predicting the 3D hand pose from monocular color image is especially difficult. To circumvent the inherent ambiguities of monocular color images, [316] uses stereo color images and estimates depth images from the stereo images, which are then used in a model-based hand pose estimation method. Recently, [325] proposed an approach that first predicts the 2D joint locations and then lifts these predictions to 3D estimates. [172] predicts 2D and 3D joint locations jointly, and then applies inverse kinematics to align these predictions. Similarly, [194] uses inverse kinematics to lift predicted 2D joint locations to 3D. [237] learns a shared embedding of images and hand poses that allows to exploit additional semi-supervised training data.

2.3 Joint Hand-Object Pose Estimation

The approaches for hand pose estimation discussed so far can only handle minor occlusions and in practice the accuracy decreases with the degree of occlusion. To specifically account for occlusions, different approaches aim at learning invariance by using images of a hand with an interacting object for training. [173, 215] simulate occlusions of the hand by adding occlusions of real objects to the training data, which, however, poses problems to acquiring such training data. [216] does the contrary: They remove the object before estimating the hand pose using a segmentation of the object. [157] uses additional temporal information to recover from occlusions.

Similar to hands in isolation, accurate methods are available for object pose estimation without occlusions [203, 293]. However, for our problem of joint hand-object pose estimation, object pose methods are required that are specifically robust to partial occlusion. Keypoint-based methods [155, 282] perform well with occlusions, but only on textured objects, and [46] requires discriminative color and features. In order to handle clutter and occlusions, local patch-based methods [25, 130], voting-based methods [57], or segmentation-based methods [84] have shown robustness to clutter and occlusions. However, they only consider objects and it is not clear how to integrate a hand in these methods.

The problem of hand and object pose estimation is much harder than considering hands or objects in isolation. A straightforward approach of combining methods of both sides does not lead to good results, since the physical constraints and visual occlusions between hand and object are not considered. Several works considered tracking of two hands or hand and object simultaneously, however, they require hands to be well-separated [251, 288], multiple-views [187], accurate segmentation [79, 87], or work at non-interactive framerates [188].

[187] considers jointly tracking a hand with an object. Therefore, they use multiple cam-

eras capturing the scene from different viewpoint to resolve ambiguities due to the occluded parts. They use a 3D model for the hand and one for the object and jointly optimize the hand and object poses using *PSO*. For the optimization, they consider edges and skin color and constrain the poses within their optimization with an interpenetration constraint. [188] considers hand-hand interactions, again by optimizing two hand models jointly using *PSO*. Therefore, they compare the rendered depth image to the input depth image. [276] uses a model-based approach for tracking hand-hand and hand-object poses. Similarly, [240] performs model-based tracking of hand and object jointly, both relying on a model for the hand and the object. These tracking-based approaches require a good initialization for each frame and are inherently prone to drifting over time.
[87] handles the occlusions of the hand due to the object by using individual pose estimators for each hand part. Further, by using a skin segmentation, they increase the uncertainty of the pose of occluded parts, and fuse the different parts together using an Markov Random Field that enforces the physical constraints. [79] relies on an accurate hand/object segmentation and masks the object from the depth image before predicting the hand pose. Their approach, however, does not consider the object pose.
For handling unknown objects, [193, 278] use an *ICP*-like object tracking that creates a model of the object on the fly. However, once it looses the temporal tracking, it fails completely.

2.4 Discussion

The methods presented in this chapter put the work done in this thesis on pose estimation into perspective of other works. The chapter should have given the reader a brief overview over related methods. A more in-depth review of the most closely related methods is given in the respective chapters.

3

Datasets and Evaluation Protocols

Contents

In this chapter we briefly introduce the datasets that we used for evaluating our methods. In order to achieve best comparison with other approaches from the literature, we resort to publicly available datasets. Further, we present the evaluation metrics for hand and object pose estimation that are used throughout the thesis to assess the accuracy of our methods.

3.1 Public Datasets

We used several publicly available datasets for our experiments and comparison with related work. An overview of different dataset characteristics is shown in Table 3.1.

The considered datasets are not only different in terms of used sensor and annotations,

Dataset	NYU [271]	MSRA [247]	ICVL [254]	BigHands [311]	STB [316]
Sensor	SL	ToF	ToF	SL	SL & color
Resolution	640×480	320×240	320×240	640×480	640×480
Distance	45-100 cm	18-56 cm	26-57 cm	21-89 cm	45-56 cm
Annotation	42 3D pos	21 3D pos	16 3D pos	21 3D pos	21 3D pos
Viewpoint	3rd	3rd	3rd	Ego/3rd	3rd
Subjects	2	9	1	10	1
Images train/test	72k/8k	76k/LOO	22k/2k	580k/295k	15k/3k

Table 3.1: Comparison of hand pose estimation datasets. *SL* denotes a structured-light sensor, and *ToF* is a time-of-flight sensor. *Ego* denotes egocentric and *3rd* third-person view. *LOO* denotes leave-one-out cross-validation.

but also different in terms of the pose space they cover. This is visualized in Figure 3.1 by using t-SNE [280]. One can clearly see that the BigHands dataset [311] covers the largest pose area, which is especially challenging for pose estimation algorithms since they need to cover the full pose space. Examples from the different datasets are shown in Figure 3.2.

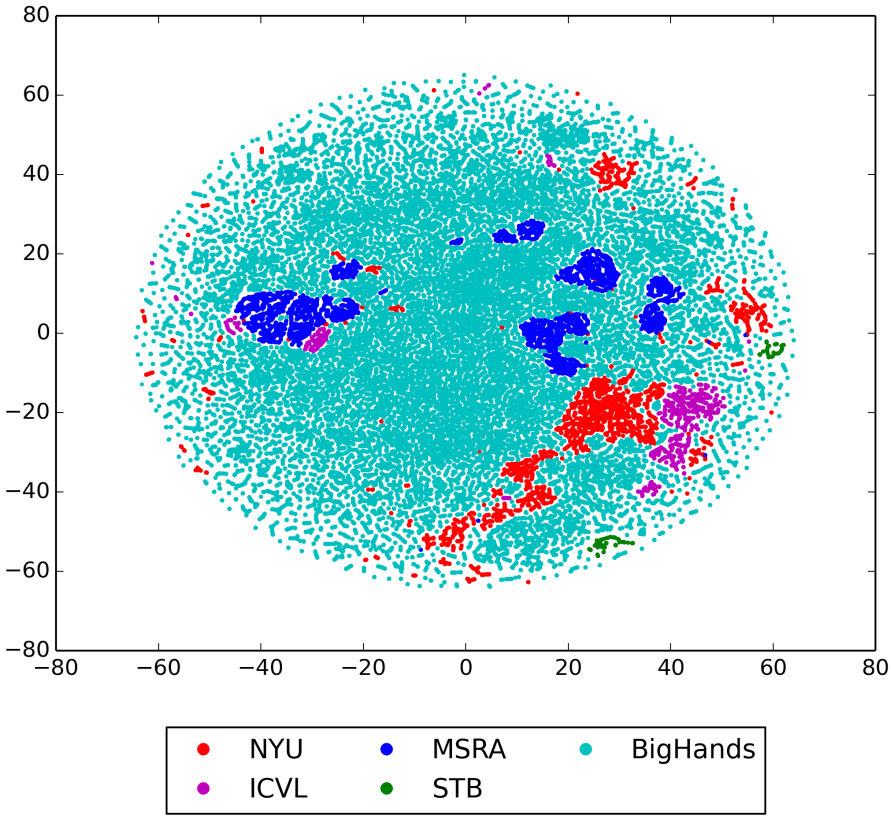


Figure 3.1: t-SNE visualization of the pose space of different datasets. The BigHands dataset [311] covers the largest pose space. The other datasets cover smaller regions within this pose space.

We would also like to note that accurate annotations for large datasets are hard to obtain. Many of the currently used datasets contain inaccurate annotations as shown in Figure 3.3. This leads to difficulties in the evaluation, since learning-based methods can overfit to the inaccurate annotations and learn the annotation bias.

3.1.1 NYU Dataset

One of the most popular benchmark datasets for 3D hand pose estimation is the NYU dataset [271]. It contains over 72k training and 8k test frames of multi-view RGB-D data. The images are captured from three different viewpoints and annotated with the 3D joint locations. The dataset was captured using the structured-light-based Primesense Carmine

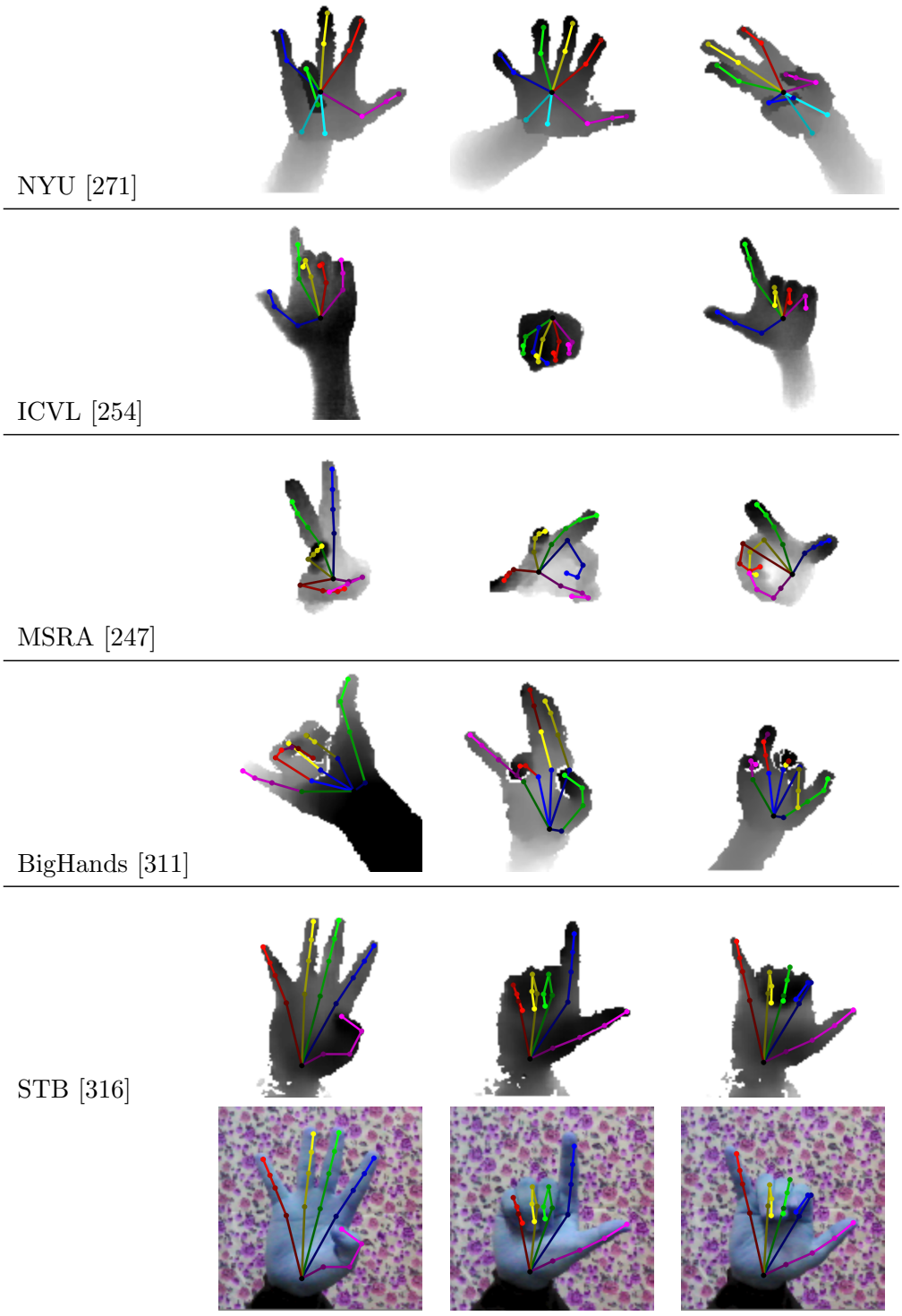


Figure 3.2: Random dataset examples for the different hand pose estimation datasets we consider in this work.

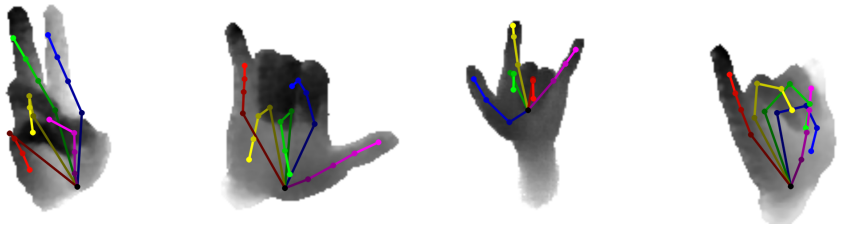


Figure 3.3: Samples with inaccurate annotations for different hand pose estimation datasets we consider in this work. Accurate annotations are hard to obtain and several datasets contain inaccurate annotations that can bias the evaluation.

1.09 sensor. Thus, the depth images show missing values as well as noisy outlines, which makes the dataset very challenging. For our experiments we use only the depth data from a single camera. Using the color images on this dataset is problematic, though, since the color pixels are projected onto the depth images causing the same artifacts in the color image than the ones present in the depth images. This renders the color images unusable for pose estimation from color images only. The dataset has accurate annotations and exhibits a high variability of different poses. The training set contains samples from a single user and the test set samples from two different users. While the ground truth contains 36 annotated joints, we follow the evaluation protocol of [183, 271] and use the same subset of 14 joints for calculating the metrics.

3.1.2 ICVL Dataset

The ICVL dataset [254] comprises a training set of over 180k depth frames showing various hand poses. The test set contains two sequences with each approximately 700 frames. The dataset is recorded using a time-of-flight Intel Creative Interactive Gesture camera and has 16 annotated joints. The depth images have a high quality with hardly any missing depth values and sharp outlines with little noise. Although the authors provide different artificially rotated training samples, we only use the genuine 22k. A drawback of this dataset is that the pose variability is limited compared to other datasets [247, 271], and annotations are rather inaccurate as discussed in [183, 249].

3.1.3 MSRA Dataset

The MSRA dataset [247] contains about 76k depth frames, each annotated with 21 3D joint locations. It was captured using a time-of-flight Creative Senz3D camera. The dataset comprises sequences from nine different subjects. We follow the common evaluation protocol [72, 248, 283] and perform a leave-one-out cross-validation: We train on eight different subjects and evaluate on the remaining subject. We repeat this procedure for each subject and report the average errors over the different runs.

3.1.4 BigHands Dataset

The BigHands dataset [311] is the largest publicly availably dataset so far, as it contains over 580k training frames and over 295k test frames. It was captured using an Intel RealSense SR300, a structured-light sensor. The dataset was automatically annotated by using six 6D magnetic sensors and inverse kinematics that results in 21 3D joint locations. However, the labels are rather inaccurate due to the used annotation process. The depth images have a high quality with hardly any missing depth values, and sharp outlines with little noise. The dataset is considered very challenging, as it has a large pose variability, and contains ten users, whereas five of them are only available for testing.

3.1.5 STB Dataset

The STB dataset [316] contains twelve sequences each containing 1500 frames of a stereo camera and an RGB-D camera. It shows a user performing diverse hand gestures in front of different backgrounds. Each image is annotated with the corresponding 3D hand pose with 21 joints. Since the STB dataset is only used for testing, learning-based method [172, 237, 325] use the synthetic RHD dataset [325] for training. It contains over 40k synthetically rendered images of humans performing various hand articulations. It consists of pairs of depth and color images, each with hand segmentation and 3D pose annotations of the same 21 joints. We slightly adjust the wrist location to better match the annotations between the two datasets.

3.1.6 DexterHO Dataset

The DexterHO dataset [240] is designed for the task of joint hand-object pose estimation. The dataset contains several sequences of RGB-D data showing different users handling an object in front of a camera, totaling over 2k frames. There are two different objects used, a large and a small cuboid. The dataset has annotations for both the hand and the object available. The hand annotation contains the 3D location of visible fingertips, and the object annotation consists of three 3D points on the object corners. Although the dataset contains color and depth, we only use the depth images for the evaluation. Examples from the dataset are shown in Figure 3.4.

3.1.7 Occlusion Datasets

As discussed in the introduction, joint hand-object pose estimation has to deal with considerable occlusions, which is an important aspect that needs to be evaluated. However, evaluating this is problematic, since, to the best of our knowledge, no dataset with 3D pose annotation of hand-held objects is available. We, therefore, resort to using more researched datasets in the context of object pose estimation, where objects occlude other objects. We consider two datasets, the Occluded LineMOD dataset [25], and the YCB-Video dataset [297]. Examples from the datasets are shown in Figure 3.4.

Occluded LineMOD Dataset The Occluded LineMOD dataset [25] consists of a sequence of 1215 frames, each frame labeled with the 6 Degrees of Freedom (DoF) poses of eight objects as well as object masks. The objects show severe occlusions, which makes pose estimation extremely challenging. The sequences were captured using an RGB-D camera with 640×480 images.

YCB-Video Dataset The recently proposed YCB-Video dataset [297] consists of 92 video sequences, where 12 sequences are used for testing and the remaining 80 sequences for training. In addition, the dataset contains 80k synthetically rendered images, which can be used for training as well. There are 21 objects in the dataset, which are taken from the YCB dataset [32] and are publicly available for purchase. The dataset is captured with two different RGB-D sensors, each providing 640×480 images. The test images are extremely challenging due to the presence of significant image noise and different illumination levels. Each image is annotated with the 6 *DoF* object poses, as well as the objects' masks.

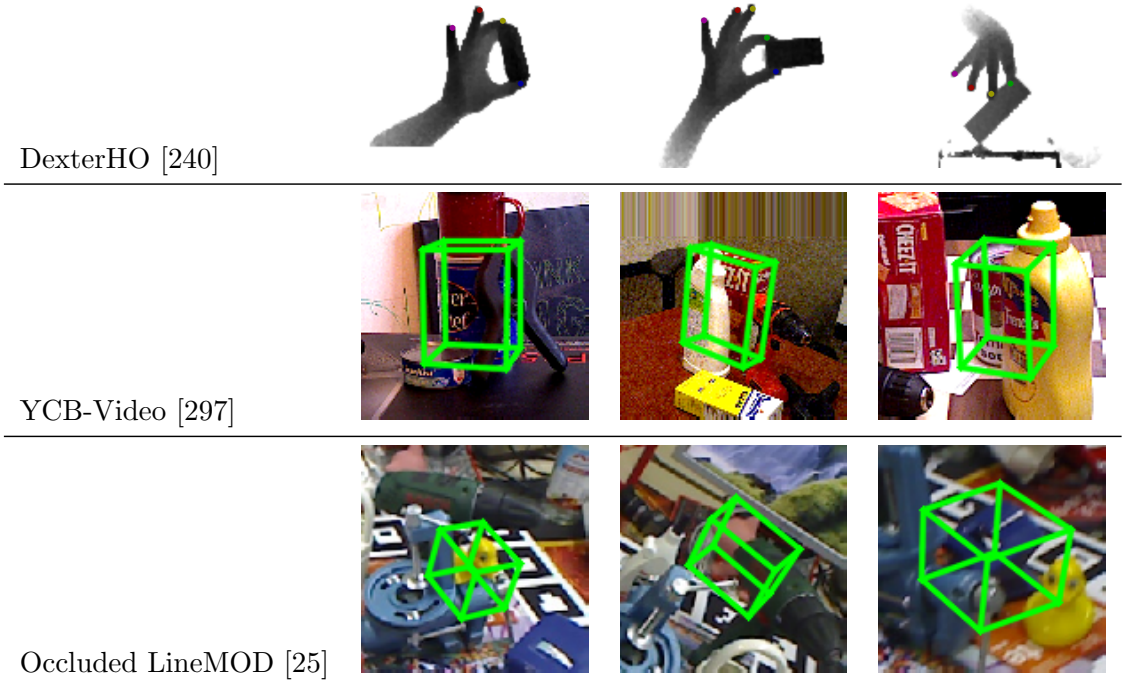


Figure 3.4: Random dataset examples for the different hand-object and occlusion datasets we consider in this work.

3.2 Evaluation Metrics

There were different evaluation metrics proposed in the literature for hand pose estimation and choosing the "correct" evaluation metric ultimately depends on the application. Also, the required pose accuracy depends on the application. While, for example, gesture recognition can tolerate errors of several centimeter when distinctive gestures as used, other applications such as virtual object manipulation in Augmented Reality (AR) or Virtual Reality (VR) might require millimeter accuracies. However, studies that systematically analyze the required levels of accuracy are yet to be published. According to Supancic *et al.* [249], a maximum joint error of 20 mm approaches the accuracy of human annotators, a maximum joint error of 50 mm seems visually consistent with a roughly correct pose estimate, and a maximum error of 100 mm appears consistent with a correct detection. Although our methods already achieve accuracies below 20 mm maximum joint error, these values can act as guidelines for assessing the severity of outliers.

Since the hand pose can be defined using different representation that we described in Section 1.1, such as joint angles or 3D joint locations, one can imagine different evaluation criteria depending on the representation. However, evaluation metrics using the Euclidean 3D joint locations are the most common criteria found in the literature. This evaluation has several advantages [253]. First, converting joint angles to 3D locations can be simply achieved by using forward kinematics, whereas the inverse is not trivial. Second, joint angles depend on the kinematic chain. For example, if the PIP angle is correctly estimated but the MCP angle is erroneous, the location of the finger tip is still wrong since the errors propagate and accumulate along the kinematic chain. Third, the global translation of hand, *i.e.*, the wrist location, is always in 3D with respect to a global coordinate system. Using this location during evaluation would result in a mixture of units which would make things complicated. Therefore, we use the 3D Euclidean error metrics throughout this thesis.

3.2.1 Per-Frame Errors

We first present the error metrics that can be applied to a single frame. We calculate the Euclidean distance between the ground truth $\mathbf{P}$ and the predicted pose $\widehat{\mathbf{P}}$. We denote the average error E_{avg} for a single frame by averaging the Euclidean distance over the number of all joints J:

$$E_{avg} = \frac{1}{J} \sum_{j=1}^{J} \|\mathbf{P}_j - \widehat{\mathbf{P}}_j\| \quad . \tag{3.1}$$

Further, the maximum error E_{max} is defined as the maximum error over all joints:

$$E_{max} = \max_{j \in [1;J]} \|\mathbf{P}_j - \widehat{\mathbf{P}}_j\| \quad . \tag{3.2}$$

This error represents a much harder metric, since a single erroneous joint can deteriorate the hand pose.

3.2.2 Set Errors

We usually deal with sequences or sets of frames that are evaluated in an overall metric. We therefore denote the poses of a set of frames as:

$$P = \{\mathbf{P}^{(n)}\}_{n=1}^{N} \quad . \tag{3.3}$$

The average error F_{avg} has established itself as the most common metric when the error should be measured as a single number for a sequence:

$$F_{avg} = \frac{1}{N} \sum_{n=1}^{N} E_{avg}(\mathbf{P}^{(n)}) \quad , \tag{3.4}$$

where the overall error is simply measured as the average over the full sequence.

However, this measure is not sensitive to outliers and [260] introduced a more challenging metric $F(\tau)$ that measures the fraction of test samples that have all predicted joints below a given maximum Euclidean distance from the ground truth:

$$F(\tau) = \frac{|\{E_{max}(\mathbf{P}) \mid E_{max}(\mathbf{P}) < \tau, \mathbf{P} \in P\}|}{|P|} \quad . \tag{3.5}$$

τ denotes a threshold that can be varied to create plots that show the fraction of frames with a worst case error below the threshold.

Another metric, the Percentage of Correct Keypoints (PCK), originating from human pose estimation [305], treats each predicted joint location separately. It measures the fraction of joints that have an Euclidean distance between the ground truth $\mathbf{P}_j$ and predicted joint location $\widehat{\mathbf{P}}_j$ below a given threshold τ:

$$PCK(\tau) = \frac{|\{\|\mathbf{P}_j - \widehat{\mathbf{P}}_j\| \mid \|\mathbf{P}_j - \widehat{\mathbf{P}}_j\| < \tau, j \in [1; J], \mathbf{P} \in P\}|}{J \cdot |P|} \quad . \tag{3.6}$$

3.2.3 Joint Hand-Object Pose Evaluation

For joint hand-object pose estimation, we need to evaluate the object pose in addition to the hand pose. There are two possibilities: (1) evaluating the hand and object pose in a combined metric, or (2) evaluating the hand pose and object pose separately in two different metric.

[240] proposed a combined metric, which combines the evaluation of the hand and the

object poses:

$$E_{HO} = \frac{1}{|V| + 1_M} \left[\sum_{i \in V} \|\mathbf{X}_i - \mathbf{G}_i\| + \frac{1_M}{3} \sum_{m \in M} \|\mathbf{Y}_m - \mathbf{O}_m\| \right] \quad . \tag{3.7}$$

The first sum measures the Euclidean distance between the predicted finger tip locations $\mathbf{X}$ and the ground truth finger tips $\mathbf{G}$ for all visible finger tips V. The second sum measures the Euclidean distance between the predicted object corners $\mathbf{Y}$ and the ground truth corners $\mathbf{O}$, where M is the set of visible corners. The object pose is only evaluated if all corners of the object are visible, *i.e.*, the indicator function $1_M = 3$ if all three corners are visible.

When using separate metrics for hand and object, we can resort to the metric for hand pose estimation introduced in the previous paragraph. For the evaluation of the 3D object pose, we use the most common metrics [26, 94, 297]: Firstly, the 2D Reprojection error. This error computes the distances between the projected 3D model points $\mathcal{M}$ in 2D image coordinates. Hereby, $\mathbf{P}$ denotes the ground truth pose, $\widehat{\mathbf{P}}$ denotes our estimated pose, and proj$(\cdot)$ denotes the projection from 3D to 2D using the intrinsic camera calibration matrix. The poses are represented as 4×4 matrices in homogeneous coordinates consisting of rotation and translation. The 2D Reprojection error can be calculated as:

$$\text{Repr} = \frac{1}{|\mathcal{M}|} \sum_{\mathbf{x} \in \mathcal{M}} \|\text{proj}(\mathbf{P} \cdot \mathbf{x}) - \text{proj}(\widehat{\mathbf{P}} \cdot \mathbf{x})\| \quad . \tag{3.8}$$

Similarly, the ADD metric calculates the average distance in 3D between the model points $\mathcal{M}$ transformed by ground truth pose $\mathbf{P}$ and the model point transformed by the estimated pose $\widehat{\mathbf{P}}$:

$$\text{ADD} = \frac{1}{|\mathcal{M}|} \sum_{\mathbf{x} \in \mathcal{M}} \|\mathbf{P} \cdot \mathbf{x} - \widehat{\mathbf{P}} \cdot \mathbf{x}\| \quad . \tag{3.9}$$

Further, for symmetric objects, the 3D distances are calculated between the closest 3D points, referred to as the ADI metric:

$$\text{ADI} = \frac{1}{|\mathcal{M}|} \sum_{\mathbf{x}_1 \in \mathcal{M}} \min_{\mathbf{x}_2 \in \mathcal{M}} \|\mathbf{P} \cdot \mathbf{x}_1 - \widehat{\mathbf{P}} \cdot \mathbf{x}_2\| \quad . \tag{3.10}$$

Part I

Fast and Accurate 3D Hand Pose Estimation

4

3D Hand Pose Estimation with Feed-forward Neural Networks

Contents

In this work, we build on the success of Convolutional Neural Networks (CNNs) and use them for their demonstrated performance. We observe, that the structure of the network is very important. We introduce and evaluate several architectures for *CNNs* to predict the 3D joint locations of a hand given a depth image. We first show that a prior on the 3D pose can be easily introduced and significantly improves the accuracy and reliability of the predictions. We name this approach *DeepPrior*. Also, we show how to use context efficiently to deal with ambiguities between fingers. These two contributions enable our method to achieve excellent results on several challenging benchmarks, both in terms of accuracy and computation times.

Since this method provides a simple and accurate approach, we further show that we can significantly increase its accuracy with other simple improvements, referred to as *DeepPrior++*: By adding Residual layers, data augmentation, and better initial hand localization, we achieve better or similar performance than more sophisticated recent methods on several challenging benchmarks while keeping the simplicity of the original method.

4.1 Introduction

Given the current trend in Computer Vision, it is natural to apply Deep Learning [223] to solve the task of 3D hand pose estimation, and a *CNN* with a standard architecture performs remarkably well when applied to this problem, as a simple experiment shows. However, the architecture of the network has a strong influence on the accuracy of the

output [43, 248] and in this chapter, we aim at identifying the architecture that performs best for this problem.

More specifically, our contribution is two-fold:

- We show that we can learn a prior model of the 3D hand pose and integrate it seamlessly to a Deep Network—denoted as *DeepPrior*—to improve the accuracy of the predicted pose. This results in a network with an unusual "bottleneck", *i.e.*, a layer with fewer neurons than the output layer.
- Like previous work [248, 272], we use a refinement stage to improve the location estimates for each joint independently. Since it is a regression problem, spatial pooling and subsampling should be used carefully for this stage. To solve this problem, we use multiple input regions centered on the initial estimates of the joints, with very small pooling regions for the smaller input regions, and larger pooling regions for the larger input regions. Smaller regions provide accuracy, larger regions provide contextual information.

We show that our original contributions lead to excellent results on several challenging benchmarks [254, 271], both in terms of accuracy and computation times. Our method runs at over 5000 frames per second (fps) on a single GPU and over 500 *fps* on a CPU, which is one order of magnitude faster than the state-of-the-art.

In 2015, an evaluation of several works on benchmark datasets [249] has shown that *DeepPrior*, the method presented in this chapter, performed state-of-the-art in terms of accuracy and speed. Since the publication of the original paper, there has been tremendous advances in the field of Machine Learning and Deep Neural Networks. We leverage recent progress in this field and update the original approach. Thus, we call the resulting approach *DeepPrior++*. Specifically:

- we update the model architecture to make the model more powerful by introducing Residual layers [91] for extracting feature maps;
- we improve the initial hand localization method. This step in DeepPrior was based on a heuristics. Here we use a trained method;
- we improve the training procedure to leverage more information from the available data.

In the remainder of this chapter, we introduce our contributions in Section 4.2 and evaluate them in Section 4.3.

4.2 Hand Pose Estimation with Deep Learning

In this section we present our original contributions to the hand pose estimation problem. We first briefly introduce the problem and a simple hand detector, which we use to get a coarse bounding box of the hand as input to the *CNN*-based pose predictors.

Then we describe our general approach that consists of two stages. For the first stage we consider different architectures that predict the locations of all joints simultaneously. Optionally, this stage can predict the pose in a lower-dimensional space, which is described next. Further, we detail the second stage, which refines the locations of the joints independently from the predictions made at the first stage. Finally, we introduce the changes we applied to obtain the DeepPrior++ method.

4.2.1 Hand Detection and Data Preprocessing

We want to estimate the 3D hand joint locations from a single depth image. We assume that a training set of depth images labeled with the 3D joint locations is available. To simplify the regression task, we first estimate a coarse 3D bounding box containing the hand using a simple method similar to [254], by assuming the hand is the closest object to the camera: We extract from the depth image a fixed-size cube centered on the center of mass of this object, and resize it to a 128×128 patch of depth values normalized to $[-1, 1]$. Points for which the depth is not available—which may happen with structured-light sensors for example—or are deeper than the back face of the cube, are assigned a depth of 1. This normalization is important for the *CNN* in order to be invariant to different distances from the hand to the camera.

4.2.2 Network Structures for Predicting the Joints' 3D Locations

We first considered two standard *CNN* architectures. The first one is shown in Figure 4.1a, and is a simple shallow network, which consists of a single convolutional layer, a max-pooling layer, and a single fully connected hidden layer. The second architecture we consider is shown in Figure 4.1b and is a deeper but still generic network [129, 272], with three convolutional layers followed by max-pooling layers and two fully connected hidden layers. All layers use Rectified Linear Unit [129] activation functions.

Additionally, we evaluated a multi-scale approach, as done for example in [63, 226, 270]. The motivation for this approach is that using multiple scales may help capturing contextual information. It uses several downscaled versions of the input image as input to the network, as shown in Figure 4.1c.

Our results will show that, unsurprisingly, the multi-scale approach performs better than the deep architecture, which performs better than the shallow one. However, our contributions, described in the next two sections, bring significantly more improvement.

4.2.3 Enforcing a Prior on the 3D Pose

So far we only considered predicting the 3D positions of the joints directly. However, given the physical constraints over the hand, there are strong correlation between the different 3D joint locations, and previous work [296] has shown that a low-dimensional embedding is sufficient to parameterize the 3D hand pose. Instead of directly predicting the 3D

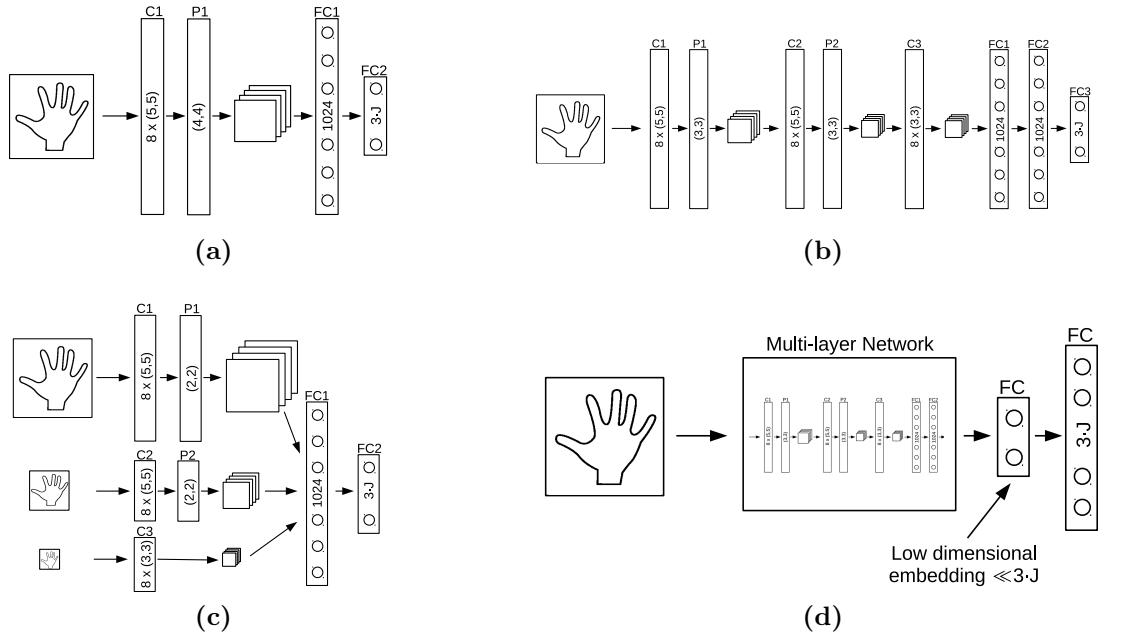


Figure 4.1: Different network architectures for the first stage. C denotes a convolutional layer with the number of filters and the filter size inscribed, FC a fully connected layer with the number of neurons, and P a max-pooling layer with the pooling size. We evaluated the performance of a shallow network (a) and a deeper network (b), as well as a multi-scale architecture (c), which was used in [63, 226]. This architecture extracts features after downscaling the input depth image by several factors. (d) All these networks can be extended to incorporate a pose prior. The shown *Multi-layer Network* can be an arbitrary neural network, with an additional layer for the prior. DeepPrior introduces a pose prior by pre-computing the weights of the last layer from a Principal Component Analysis (PCA) applied to the 3D hand pose data. This causes an unusual bottleneck with less neurons than the output layer.

joint locations, we can therefore predict the parameters of the pose in a lower-dimensional space. As this enforces constraints of the hand pose, it can be expected that it improves the reliability of the predictions, which will be confirmed by our experiments.

As shown in Figure 4.1d, we implement the pose prior into the network structure by introducing a "bottleneck" in the last layer. This bottleneck is a layer with less neurons than necessary for the full pose representation, *i.e.*, $\ll 3 \cdot J$ for J 3D joints. It forces the network to learn a low-dimensional representation of the training data, that implements the physical constraints of the hand. Similar to [296], we rely on a linear embedding. The embedding is enforced by the bottleneck layer and the reconstruction from the embedding to pose space is integrated as a separate hidden layer added on top of the bottleneck layer. The weights of the reconstruction layer are set to compute the back-projection into the $3 \cdot J$-dimensional 3D joint space. The resulting network therefore directly computes the full pose. We initialize the reconstruction weights with the major components from a *PCA* of the 3D hand pose data and then train the full network using back-propagation. Using

this approach we train the networks described in the previous section.

The embedding can be as small as 8 dimensions for a 42-dimensional pose vector to fully represent the 3D pose as we show in the experiments.

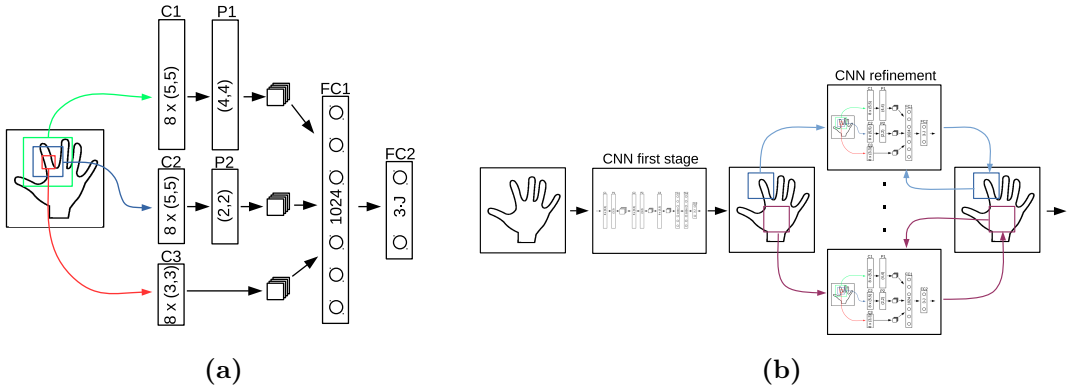


Figure 4.2: Architecture for refining the joint locations during the second stage. We use a different network for each joint in order to refine its location estimate as provided by the first stage. (a) The architecture we propose uses overlapping inputs centered on the joint to refine. Pooling with small regions is applied to the smaller inputs, while the larger inputs are pooled with larger regions. The smaller inputs allow for higher accuracy, the larger ones provide contextual information. We experimentally show that this architecture is more accurate than a more standard network architecture. (b) shows a generic architecture of an iterative refinement, where the output of the previous iteration is used as input for the consecutive iteration. As for Figure 4.1, C denotes a convolutional layer, FC a fully connected layer, and P a max-pooling layer.

4.2.4 Refining the Joint Location Estimates

The first stage, presented in the previous section, provides estimates for the locations of all the joints simultaneously. As done in [248, 272], these estimates can then be refined independently.

Spatial context is important for this refinement step to avoid confusion between the different fingers. The best performing architecture we experimented with is shown in Figure 4.2a. We will refer to this architecture as *ORRef* for *Refinement with Overlapping Regions*. It uses as input several patches of different sizes but all centered on the joint location predicted by the first stage. No pooling is applied to the smallest patch, and the size of the pooling regions then increases with the size of the patch. The larger patches provide more spatial context, whereas the absence of pooling on the small patch enables better accuracy.

We also considered a standard *CNN* architecture as a baseline, represented in Figure 4.1b, which relies on a single input patch. We will refer to this baseline as *StdRef* for *Refinement with Standard Architecture*.

To further improve the accuracy of the location estimates, we iterate this refinement step several times, by centering the network on the location predicted at the previous iteration.

4.2.5 DeepPrior++

In this section, we describe our changes to improve the original DeepPrior approach. Our improved method—referred to as DeepPrior++—relies on the same original method presented in Section 4.2, but implements novel insights and progress gathered within two years in the fields of Machine Learning and Deep Learning. It is interesting to see, that we can achieve huge improvements by using the same original method, only extended with a better implementation and applying new Deep Learning techniques.

4.2.5.1 Improved Training Data Augmentation

Since our approach is data-driven, we aim at leveraging as much information as possible from the available data. There have been many different augmentation methods used in literature [129, 271], such as scaling, flipping, mirroring, rotating, *etc*. In this chapter we use depth images, which give rise to specific data augmentation methods. Specifically, we use rotation, scaling, and translation, as well as different combinations of them.

Rotation The hand can be rotated easily around the forearm. This rotation can be approximated by simple in-plane rotation of the depth images. We use random in-plane rotations of the image and change the 3D annotations accordingly by projecting the 3D annotations onto the 2D image, applying the same in-plane rotation, and projecting the 2D annotations back to 3D coordinates. The rotation angle is sampled from a uniform distribution with the interval $[-180°, 180°]$.

Scaling The MSRA [247] and NYU [271] datasets contain different persons, with different hand size and shape. Although DeepPrior is not explicitly invariant to scale, we can train the network to be invariant to the hand size by varying the size of the crop in the training data. Therefore, we scale the 3D bounding box for the crop from the depth image by a random factor sampled from a normal distribution with mean of 1 and variance of 0.02. This changes the appearance of the hand size in the cropped cube and we scale the 3D joint locations according to the random factor.

Translation Since the hand 3D localization is not perfect, we augment the training set by adding random 3D offsets to the hand 3D location, and center the crops from the depth images on these 3D locations. We sample the random offsets from a normal distribution with a variance of 5 mm, which is comparable to the error of the hand 3D detector we use. We also modify the 3D annotations according to this offset.

Online Augmentation The augmentation is performed online during training and thus the network sees different samples at each epoch. This leads to more than 10M different samples in total. The augmentation helps to prevent overfitting and to be more robust to deviations of the hand from the training set. Although the samples are correlated, it significantly helps at test time, as we show in the experiments.

Robust Prior Similarly, we also improve the prior, which is obtained by applying *PCA* to the 3D hand poses. We sample 1M poses, by randomly using rotation, scaling, and translation of the original poses in 3D. We use this augmented set of 3D poses for calculating the prior.

4.2.5.2 Refined Hand Localization

The original DeepPrior used a very simple hand detection. It was based on the center of mass of the depth segmentation of the hand. Therefore, the hand was segmented using depth-thresholding and the 3D center of mass was calculated. Then, a 3D bounding box was extracted around the center of mass.

DeepPrior++ still uses this method but introduces a refinement step that significantly improves the final accuracy. This refinement step relies on a regression *CNN*. This *CNN* is applied to the 3D bounding box centered on the center of mass, and is trained to predict the location of the metacarpophalangeal (MCP) joint of the middle finger, which we use as referential. We also use augmented training data to train this *CNN* as described in Section 4.2.5.1.

For real-time applications, instead of extracting the center of mass from each frame, we apply this regression *CNN* to the hand location of the previous frame. This remains accurate while being faster in practice.

4.2.5.3 More Powerful Network Architecture

Residual Networks Since the introduction of DeepPrior, there has been much research on better deep architectures [92, 232, 250], and the Residual Network (ResNet) architecture [92] appears to be one of the best performing models.

Our model is similar to the 50-layer ResNet model of [92]. Since ResNet was originally proposed for image classification, we adapt the architecture to fit our regression problem. Most importantly, we remove the Global Average Pooling, and add two fully connected layers. The input to the network is 128×128 px, with values normalized to $[-1, 1]$. The adapted ResNet model is shown in Figure 4.3. The network contains an initial convolution layer with 64 filters and 2×2 max-pooling. This convolutional layer is followed by four residual modules, each with a stride of 2×2, and with $\{64, 128, 256, 256\}$ filters.

The much simpler model used for refining the hand localization is shown in Figure 4.4. It consists of three convolutional layers with max-pooling, and two fully connected layers with Dropout [243].

Regularization using Dropout The ResNet model can overfit, and we experienced this behavior especially on datasets with small hand pose variation [254]. Therefore, we introduce Dropout [243] to the model, which was shown to provide an effective way of regularizing a neural network. We apply binary Dropout with a dropout rate of 0.3 on both fully connected layers after the residual modules. This enables training high capacity ResNet models while avoiding overfitting and achieving highly accurate predictions.

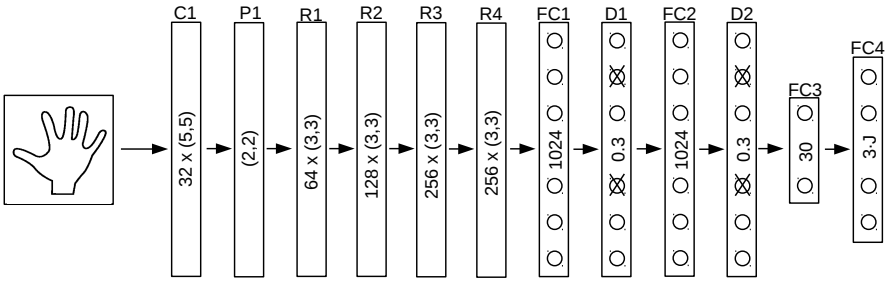


Figure 4.3: Our ResNet architecture. C denotes a convolutional layer with the number of filters and the filter size inscribed, FC a fully connected layer with the number of neurons, D a Dropout layer with the probability of dropping a neuron, R a residual module with the number of filters and filter size, and P a max-pooling layer with the pooling region size. The hand crop from the depth image is fed to the ResNet that predicts the final 3D hand pose.

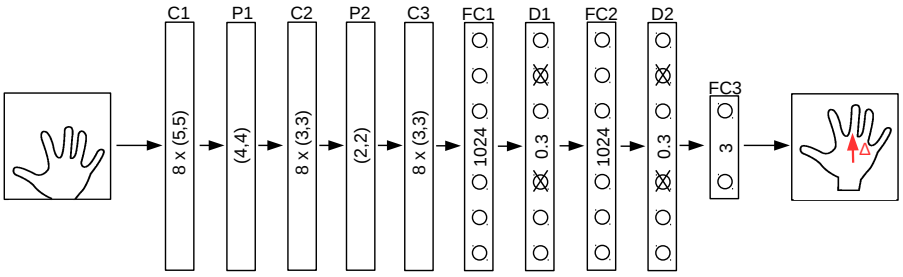


Figure 4.4: Network architecture for refining hand localization. As in Figure 4.3, C denotes a convolutional layer, FC a fully connected layer, D a Dropout layer, and P a max-pooling layer. The initial hand crop from the depth image is fed to the network that predicts an offset to correct an inaccurate hand localization.

4.3 Evaluation

In this section we evaluate the different architectures introduced in the previous section on several challenging benchmarks. We first describe the network optimization and further present the results, quantitatively as well as qualitatively, on several challenging benchmark dataset.

4.3.1 Meta-Parameters and Optimization

The performance of neural networks depends on several meta-parameters, and we performed a large number of experiments varying the meta-parameters for the different architectures we evaluated. We report here only the results of the best performing sets of meta-parameters for each method. However, in our experiments, the performance depends more on the architecture itself than on the values of the meta-parameters.

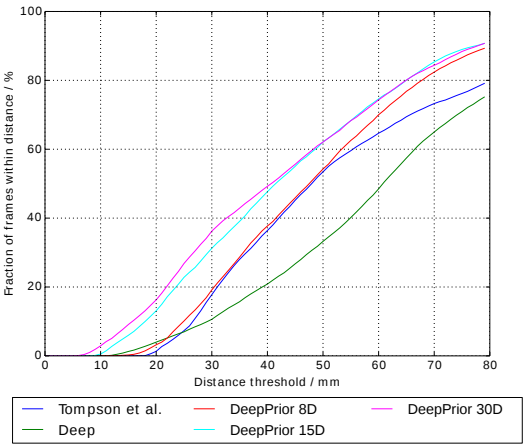
We trained the different architectures by minimizing the distance between the prediction and the expected output per joint, and a regularization term for weight decay to prevent over-fitting, where the regularization factor is 0.001. We do not differ between occluded and non-occluded joints. Because the annotations are noisy, we use the robust Huber loss [104] to evaluate the differences. We optimize the network parameters using the gradient descent algorithm ADAM [125] with standard hyper-parameters, a batch size of 128, and a learning rate of 0.0001 for 100 epochs.

4.3.2 Importance of the Pose Prior

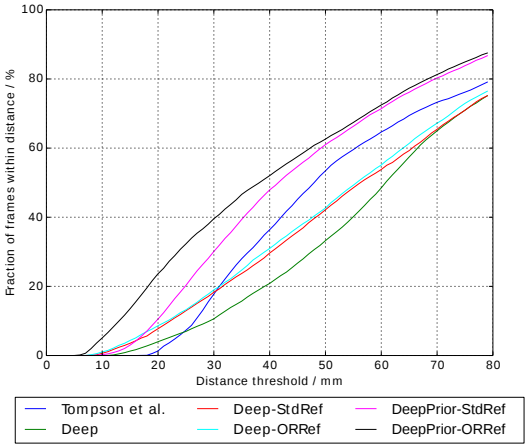
In Figure 4.5a and 4.5c we compare different embedding dimensions and direct regression in the full $3 \cdot J$-dimensional pose space for the NYU dataset [271] and the ICVL dataset [254], respectively. The evaluation on both datasets shows that enforcing a pose prior is beneficial compared to direct regression in the full pose space. Only 8 dimensions out of the original 42- or 48-dimensional pose spaces are already enough to capture the pose and outperform the baseline on both datasets. However, the 30-dimensional embedding performs best, and thus we use this for all further evaluations. The results on the ICVL dataset, which has noisy annotations, are not as significant, but still consistent with the results on the NYU dataset.

The baseline on the NYU dataset of Tompson *et al.* [271] only provide the 2D locations of the joints. For comparison, we follow their protocol and augment their 2D locations by taking the depth of each joint directly from the depth images to derive comparable 3D locations. Depth values that do not lie within the hand cube are set to the ground truth depth value to avoid large errors. This protocol, however, has a certain influence on the error metric, as evident in Figure 4.7a. The augmentation works well for some joints, as apparent by the average error. However, it is unlikely that the augmented depth is correct for all joints of the hand, *e.g.*, the 2D joint location lies on the background or is self-occluded, thus causing higher errors for individual joints. When using the evaluation metric of [260], where all joints have to be within a maximum distance, this outlier has a strong influence, in contrast to the evaluation of the average error, where an outlier can be insignificant for the mean. Thus we outperform the baseline more significantly for the distance threshold than for the average error.

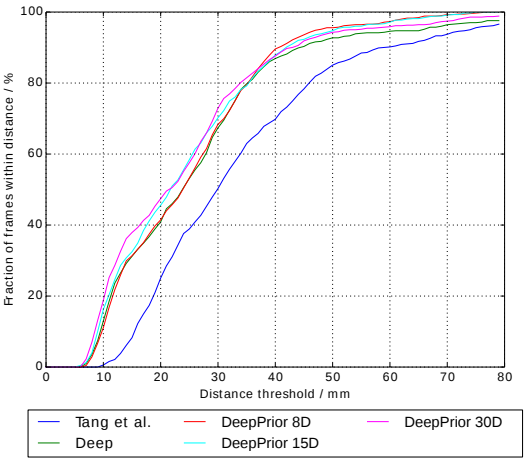
We further show the average error over the embedding dimensions in Figure 4.6. We compare the accuracy of DeepPrior to a Deep Network without prior. The accuracy of DeepPrior for very low-dimensional embeddings is worse than the network without prior,



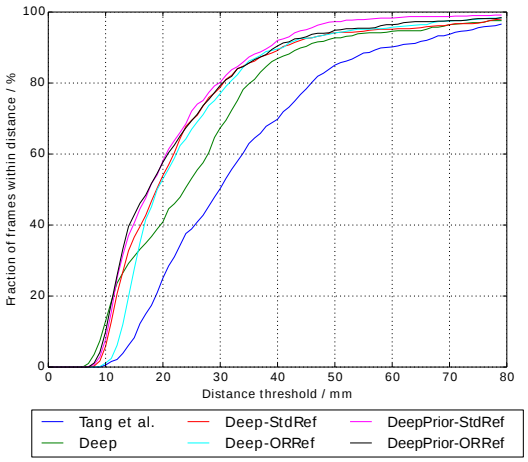
(a) Pose prior on NYU dataset.



(b) Refinement on NYU dataset.



(c) Pose prior on ICVL dataset.



(d) Refinement on ICVL dataset.

Figure 4.5: Importance of the pose prior (left) and the refinement stage (right). We plot the fraction of frames where all joints are within a maximum distance for different approaches. A higher area under the curve denotes more accurate results. **Left (a), (c):** We show the influence of the dimensionality of the pose embedding. The optimal value is around 30, but using only 8 dimensions performs already very well. The pose prior significantly increases the accuracy, even without the refinement step. **Right (b), (d):** We show that our architecture with overlapping input patches, denoted by the *ORRef* suffix, provides higher accuracy for refining the joint positions compared to a standard deep *CNN*, denoted by the *StdRef* suffix. For the baseline of Tompson *et al.* [271] we augment their 2D joint locations with the depth from the depth images, as done by [271], and depth values that do not lie within the hand cube are set to the ground truth depth value to avoid large errors.

since the embedding is not expressive enough to represent all poses. For embeddings with more than 10 dimensions, the accuracy of DeepPrior is better than the network without prior. This approximately corresponds to a cumulative explained variance of the *PCA* components of more than 95%. For embedding dimensions larger than 30, the error does not change significantly and is within the standard deviation of the different network initializations.

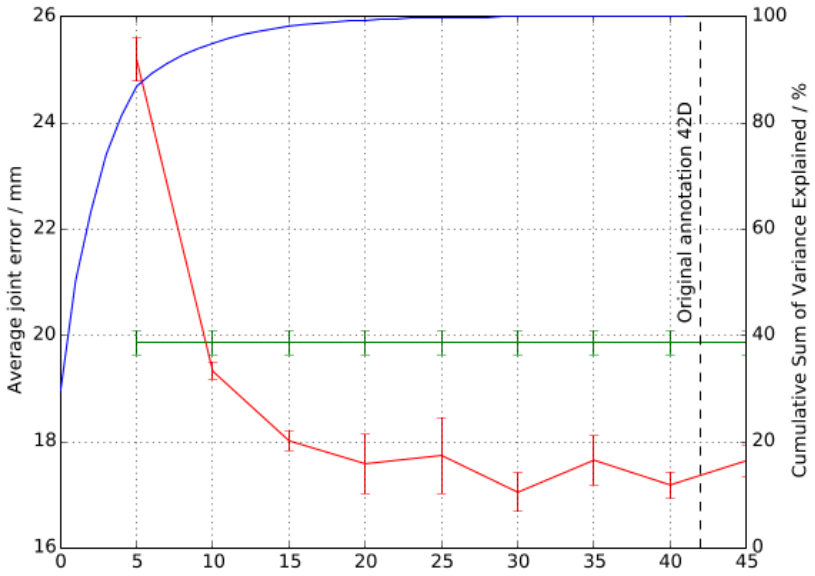


Figure 4.6: Accuracy over embedding dimensions. We plot the average 3D joint error with standard deviations for five differently initialized networks on the NYU dataset [271]. The red curve shows the accuracy for DeepPrior with different embedding dimensions, and the green curve shows the accuracy for a Deep Network without prior. We further plot the cumulative explained variance of the different number of *PCA* components in blue.

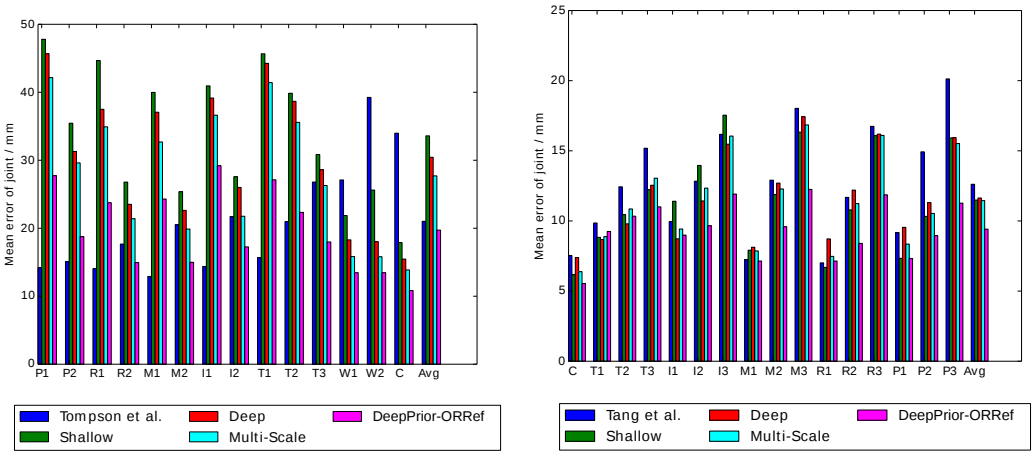
4.3.3 Increasing Accuracy with Pose Refinement

The refinement stage can be used to further increase the location accuracy of the predicted joints. We achieved the highest accuracy by using our *CNN* with pose prior as first stage, and then applying the second iterative refinement stage with our *CNN* with overlapping input patches, denoted *ORRef*.

The results in Figure 4.5b, 4.5d and 4.7 show that applying the refinement improves the location accuracy for different base *CNNs*. From rather inaccurate initial estimates, as provided by the standard deep *CNN*, our proposed ORRef performs only slightly better than refinement with the baseline deep *CNN*, denoted by *StdRef*. This is because for large initial errors only the larger input patch provides enough context for reasoning about the offset. The smaller input patch cannot provide any information if the offset is bigger than

the patch size. For more accurate initial estimates, as provided by our deep *CNN* with pose prior, the ORRef takes advantage from the small input patch which does not use pooling for higher accuracy. We iterate our refinement two times, since iterating more often does not provide any further increase in accuracy.

We would like to emphasize that our results on the ICVL dataset, with an average accuracy below 10 mm, already scratch at the uncertainty of the labeled annotations. As already mentioned, the ICVL dataset suffers from inaccurate annotations, as we show in some qualitative samples in Figure 4.8 first and fourth column. While this has only a minor effect on training, the evaluation is more affected. We evaluated the accuracy of the test sequence by revising the annotations in image space and calculated an average error of 2.4 mm with a standard deviation of 5.2 mm.



(a) NYU dataset. **(b)** ICVL dataset.

Figure 4.7: Evaluation of average 3D joint errors. For completeness and comparability we show the average joint errors, which are, however, not as decisive as the evaluation in Figure 4.5. Though, the results are consistent. The evaluation of the average error is more tolerant to larger errors of a single joint, which deteriorate the pose as for Figure 4.5, but are insignificant for the mean if the other joints are accurate. Our proposed architecture *DeepPrior-ORRef*, the constrained pose *CNN* with refinement stage, provides the highest accuracy. For the ICVL dataset, the simple baseline architectures already outperform the baseline. However, they cannot capture the higher variations in pose space and noisy images of the NYU dataset, where they perform much worse. The palm and fingers are indexed as C: palm, T: thumb, I: index, M: middle, R: ring, P: pinky, W: wrist.

4.3.4 Runtime

Table 4.1 provides a comparison of the runtimes of the different methods, both on CPU and GPU. They were measured on a computer equipped with an Intel Core i7, 16GB of RAM, and an nVidia GeForce GTX 780 Ti GPU. Our methods are implemented in Python

Architecture	GPU	CPU
Shallow	0.07 ms	1.85 ms
Deep [129]	0.1 ms	2.08 ms
Multi-Scale [63]	0.81 ms	5.36 ms
DeepPrior	0.09 ms	2.29 ms
Refinement	2.38 ms	62.91 ms
Tompson *et al.* [271]	5.6 ms	-
Tang *et al.* [254]	-	16 ms

Table 4.1: Comparison of different runtimes. Our *CNN* with pose prior (*DeepPrior*) is faster by a magnitude compared to the other methods when considering pose estimation only. We can further increase the accuracy using the refinement stage, still at competitive speed. All of the denoted baselines use state-of-the-art hardware comparable to ours.

using Theano [15], which offers an option to select between the CPU and the GPU for evaluating *CNNs*. Our different models perform very fast, up to over 5000 *fps* on a single GPU. Training takes about five hours for each *CNN*. The Deep Network with pose prior performs very fast and outperforms all other methods in terms of accuracy. However, we can further refine the joint locations at the cost of higher runtime.

4.3.5 Qualitative Results

We present qualitative results in Figure 4.8. The typical problems of structured-light-based sensors, such as missing depth, can be problematic for accurate localization. However, only partially missing parts, as shown in the third and fourth columns for example, do not significantly deteriorate the result. The location of the joint is constrained by the learned hand model. If the missing regions get too large, as shown in the fifth column, the accuracy gets worse. However, because of the use of the pose prior, the predicted joint locations still preserve the learned hand topology. The erroneous annotations of the ICVL dataset deteriorate the results, as our predicted locations during the first stage are sometimes more accurate than the ones obtained during the second stage, as evident for example on the pinky in the first or fourth column.

4.3.6 DeepPrior++

We evaluate our DeepPrior++ approach on three public benchmark datasets for hand pose estimation: the NYU dataset [271], the ICVL dataset [254], and the MSRA dataset [247]. We further present additional results on the more recent BigHands dataset [311], and show the generalizability of our method to color images on the STB dataset [316]. For the comparison with other methods, we focus here on works that were published after the original DeepPrior paper. There are different evaluation metrics used in the literature for

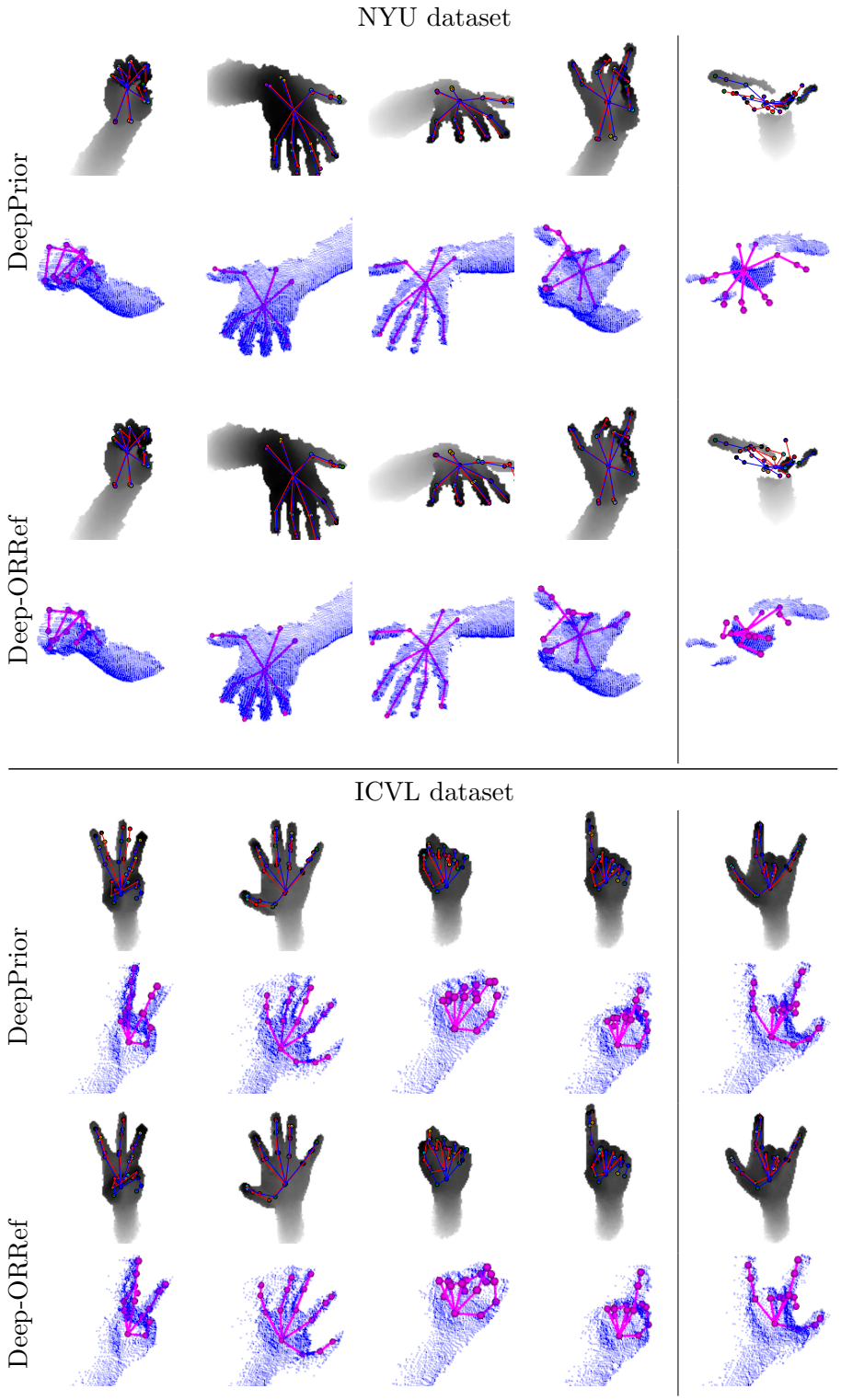


Figure 4.8: Qualitative results. We show the inferred joint locations on the depth images (in gray-scale), as well as the 3D locations with the point cloud of the hand (blue images) rendered from a different viewpoint. The ground truth is shown in blue, our results in red. The point cloud is only annotated with our results for clarity. The right column shows erroneous results. One can see the difference between the pose prior and the local refinement, especially in the presence of missing depth values as shown in the fifth column. While the pose prior still preserves the hand topology, the local refinement cannot reason about the locations in the presence of missing depth data.

Method	Average 3D error
DeepPrior (Chapter 4)	19.8 mm
Feedback (Chapter 5)	16.2 mm
Deng *et al.* [52] (Hand3D)	17.6 mm
Guo *et al.* [82] (REN)	13.4 mm
Bouchacourt *et al.* [22] (DISCO)	20.7 mm
Zhou *et al.* [321] (DeepModel)	16.9 mm
Xu *et al.* [300] (Lie-X)	14.5 mm
Neverova *et al.* [178]	14.9 mm
Wan *et al.* [283] (Crossing Nets)	15.5 mm
Fourure *et al.* [67] (JTSC)	16.8 mm
Zhang *et al.* [315]	18.3 mm
Madadi *et al.* [156]	15.6 mm
DeepPrior++	**12.3 mm**

Table 4.2: Comparison with state-of-the-art on the NYU dataset [271]. We report the average 3D error. DeepPrior++ significantly performed better than all other methods for this dataset.

hand pose estimation, and we report the numbers stated in the papers or measured from the graphs if provided, and/or plot the relevant graphs for comparison.

For all experiments, we report the results for a 30-dimensional *PCA* prior. By using an efficient implementation for data augmentation, the training time is the same for all experiments, approximately 10 hours on a computer with an Intel i7 with 3.2GHz and 64GB of RAM, and an nVidia GTX 980 Ti graphics card.

4.3.6.1 NYU Dataset

Our results are shown in Table 4.2 together with a comparison to state-of-the-art methods. We compare DeepPrior++ to several related methods, and it significantly outperforms the other methods.

In Figure 4.9 we compare our method with other discriminative approaches. Although Supancic *et al.* [249] report very accurate results for a small fraction of the frames, our approach significantly performs better for the majority of the frames.

In Figure 4.10 we compare state-of-the-art methods using a different evaluation protocol, *i.e.*, we follow the protocol of [251, 259], who evaluate the first 2400 frames of the test set. Also for this protocol, we significantly outperformed the state-of-the-art method of Taylor *et al.* [259]. Note that [251, 252, 259] require a, possibly user-specific, 3D hand model, whereas our method only uses training data without any 3D model. Comparing to these recent approaches, our method is easier and faster to train, has a simpler architecture, is more accurate, and runs at a comparable speed, *i.e.*, real-time.

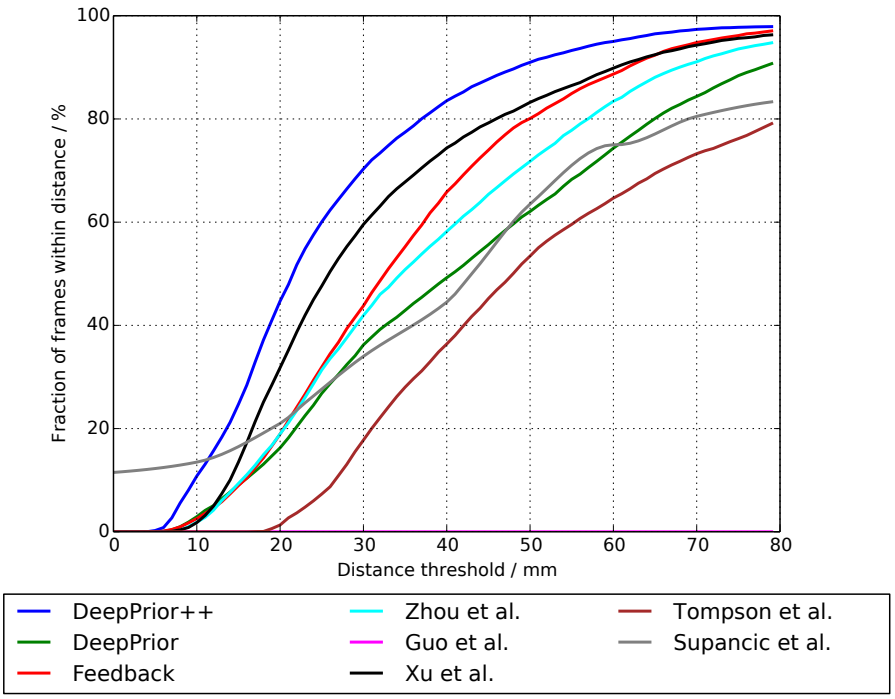


Figure 4.9: Comparison with state-of-the-art discriminative methods on the NYU dataset [271]. We plot the fraction of frames where all joints are within a maximum distance from the ground truth. A larger area under the curve indicates better results. Our proposed approach performed best among other discriminative methods.

4.3.6.2 ICVL Dataset

We show a comparison to different state-of-the-art methods in Table 4.3. However, the gap to other methods is much smaller. This may be attributed to the fact that the dataset is much easier, with smaller pose variations [249], and due to errors in the annotations for the evaluation [183, 249].

In Figure 4.11 we compare DeepPrior++ to other methods on the ICVL dataset [254]. Our approach performs similar to the works of Guo *et al.* [82], Wan *et al.* [285], and Tang *et al.* [255], all achieving state-of-the-art accuracy on this dataset. This might be an indication that the performance on the dataset is saturating, and the remaining error is due to the annotation uncertainty. This empirical finding is similar to the discussion in [249]. Although Tang *et al.* [255] performs slightly better in some parts of the curve in Figure 4.11, our approach performs significantly better on the NYU dataset, as shown in Figure 4.10.

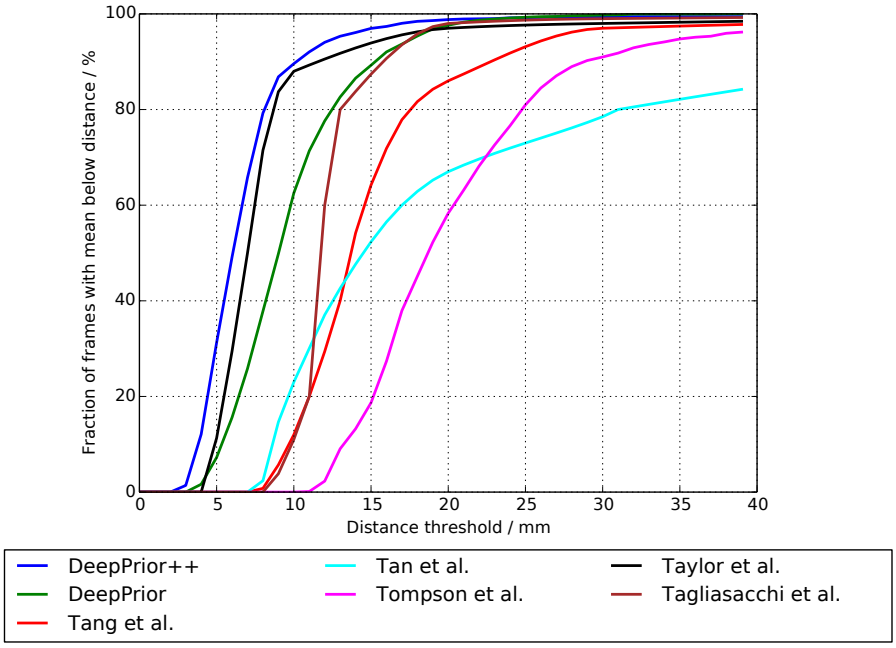


Figure 4.10: Comparison with state-of-the-art model-based methods on the NYU dataset [271]. We plot the fraction of frames where the average joint error per frame is within a maximum distance from the ground truth, following the protocol of [251, 259]. A larger area under the curve indicates better results. Our proposed approach even outperformed model-based approaches on this dataset, with more than 90% of the frames having an error smaller than 10 mm.

Method	Average 3D error
DeepPrior (Chapter 4)	10.4 mm
Deng *et al.* [52] (Hand3D)	10.9 mm
Tang *et al.* [254] (LRF)	12.6 mm
Wan *et al.* [285]	8.2 mm
Zhou *et al.* [321] (DeepModel)	11.3 mm
Sun *et al.* [247] (HPR)	9.9 mm
Wan *et al.* [283] (Crossing Nets)	10.2 mm
Fourure *et al.* [67] (JTSC)	9.2 mm
Krejov *et al.* [128] (CDO)	10.5 mm
DeepPrior++	**8.1 mm**

Table 4.3: Comparison with state-of-the-art on the ICVL dataset [254]. We report the average 3D error.

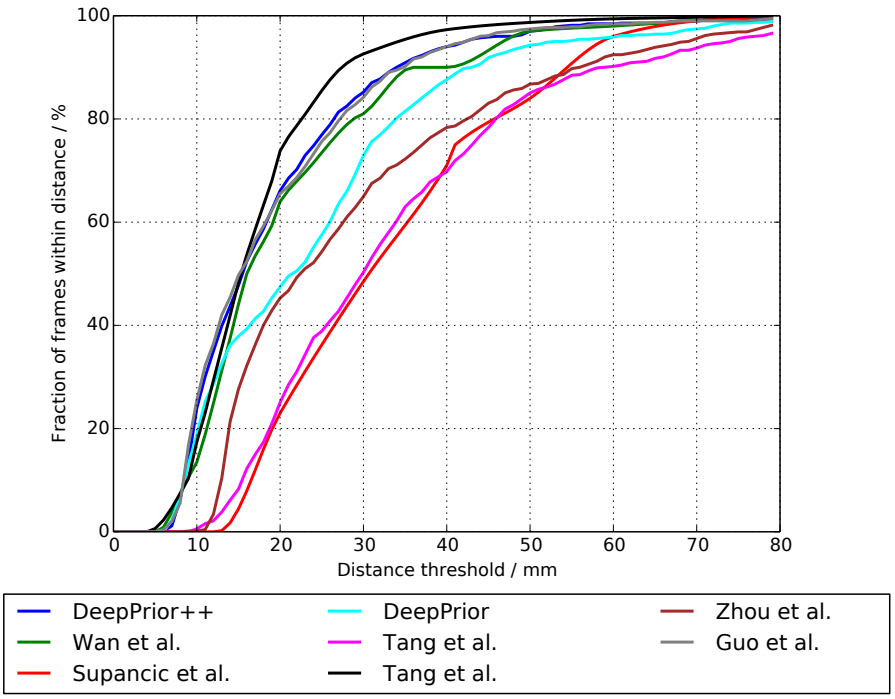


Figure 4.11: Comparison with state-of-the-art on the ICVL dataset [254]. We plot the fraction of frames where all joints are within a maximum distance from the ground truth. Several works show a similar error curve, which can be an indicator for saturating performance for this dataset.

Method	Average 3D error
Ge *et al.* [72]	13.2 mm
Sun *et al.* [247] (HPR)	15.2 mm
Wan *et al.* [283] (CrossingNets)	12.2 mm
Yang *et al.* [303]	13.7 mm
DeepPrior++	**9.5 mm**

Table 4.4: Comparison with state-of-the-art on the MSRA dataset [247]. We report the average 3D error averaged over all folds of the leave-one-out cross-validation. DeepPrior++ significantly performed better than all other methods for this dataset.

4.3.6.3 MSRA Dataset

A comparison of the average 3D error is shown in Table 4.4. We perform leave-one-out cross-validation by using one subject for testing and the remaining subjects for training. Again, DeepPrior++ outperforms the existing methods by a large margin of 3 mm. In Figure 4.12, DeepPrior++ also outperforms all other methods on the plotted metric, which shows that it is also able to handle different users' hands.

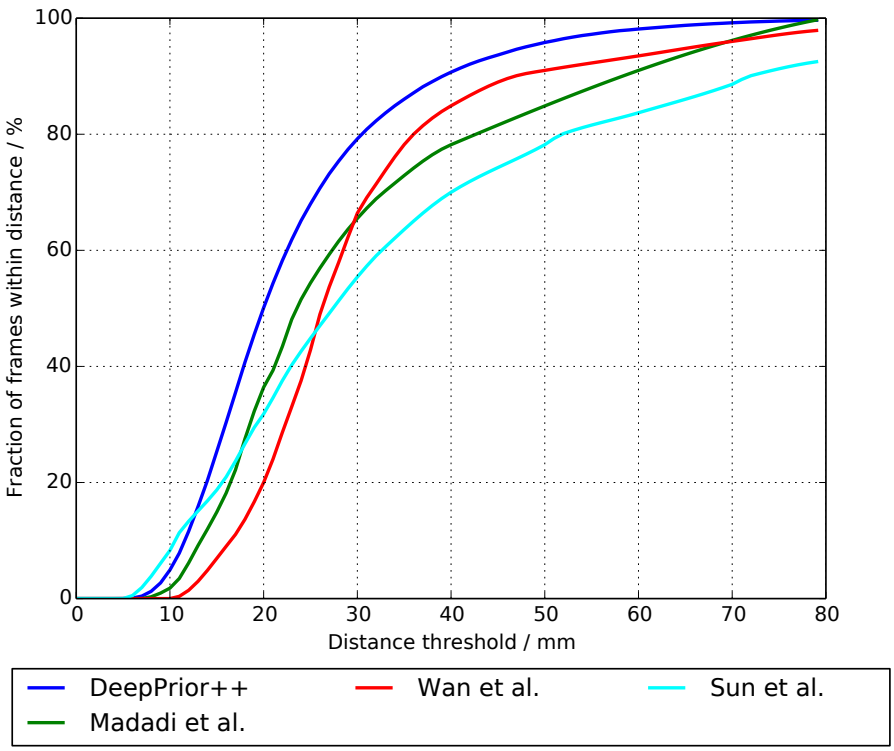


Figure 4.12: Comparison with state-of-the-art on the MSRA dataset [247]. We plot the fraction of frames where all joints are within a maximum distance from the ground truth. A larger area under the curve indicates better results. Our approach significantly outperformed state-of-the-art discriminative approaches on this dataset.

4.3.6.4 STB Dataset

We further present an evaluation of our method for 3D pose estimation from color images on the STB dataset [316]. Our method can be trained on different data modalities without modifications, which shows the general applicability of our approach. For the evaluation on the STB dataset, we also used the synthetic RHD dataset [325] for training as done in [172, 325].

A quantitative evaluation is shown in Table 4.5 using the average 3D End Point Error (EPE) metric, which is the *de facto* standard metric for evaluation on the STB dataset. We achieve very accurate results compared to other state-of-the-art methods. Since the dataset contains color and depth images, we trained and tested our method on color and depth images separately. A comparison of the results trained with different modalities shows that it is easier to infer the 3D pose from depth images, resulting in lower errors. The results from color images are still competitive.

We show qualitative results in Figure 4.13. Note, that although we use monocular color images, we estimate 3D joint locations that are subsequently projected to the image

Method	Avg. 3D *EPE*
Zimmermann&Brox [325]	12.21 mm
Müller *et al.* [172] (GANerated)	13.22 mm
DeepPrior++ color	11.07 mm
DeepPrior++ depth	8.16 mm

Table 4.5: Comparison with state-of-the-art on the STB dataset [316]. We report the average 3D *EPE* for 3D hand pose estimation from monocular color images. For comparison, we also report the accuracy of DeepPrior++ using depth images. For the method of [172] we report the accuracy on a per-frame basis without the temporal constraints used in their paper.

for visualization.

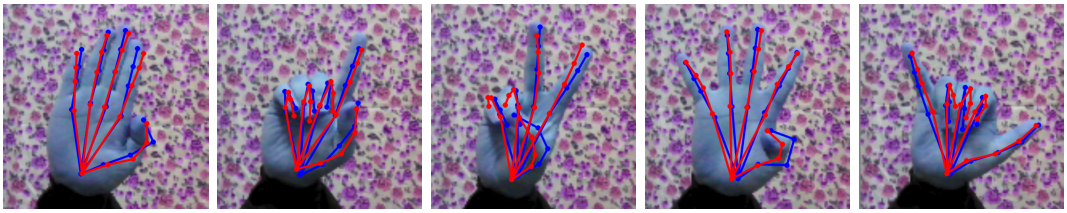


Figure 4.13: Qualitative results on the STB dataset [316]. We show the inferred 3D joint locations projected on the color images. Ground truth is shown in blue, the predicted poses in red.

4.3.6.5 BigHands Dataset

We also evaluate our method on the BigHands dataset [311]. This dataset is very challenging due to the mixed egocentric and third person viewpoints and due to multiple subjects. Due to the annotation process there are considerable annotation errors present in the dataset. A quantitative evaluation is shown in Table 4.6[1]. Note that the top performing methods were published after our work, but still our method can achieve competitive results. The top performing methods [73, 169, 170] use 3D *CNNs* with voxel-based representations. This can be advantageous since they can directly represent the 3D point measurements from the depth sensor in the voxel grid, whereas we represent these measurements as intensities in a 2D image. Although these voxel representations results in more powerful architectures, they are computationally inefficient as shown in [213], since most input voxels are filled with zeros.

Qualitative results are shown in Figure 4.14. Our method provides accurate results for third person and egocentric viewpoints.

[1]The results of the other methods were take from [310] and from the leaderboard of the dataset `https://competitions.codalab.org/competitions/17356` as of 2018-08-08.

Method	Average 3D error
Moon *et al.* [170]†	9.95 mm
Molchanov *et al.* [100, 169]†	9.97 mm
Ge *et al.* [73]‡	11.30 mm
Chen *et al.* [39]‡	11.70 mm
Yang *et al.* [302]	16.61 mm
Wan *et al.* [283]	17.78 mm
Akiyama *et al.* [1]	16.61 mm
Madadi *et al.* [156]	14.74 mm
Li *et al.* [143]	12.79 mm
DeepPrior++	12.94 mm

Table 4.6: Comparison with state-of-the-art on the BigHands dataset [311]. We report the average 3D error. Note that the top performing methods were published after our work (ICCVW'17), denoted as † CVPR'18 and ‡ preprint'18.

Figure 4.14: Qualitative results on the BigHands dataset [311]. We show the inferred 3D joint locations projected on the depth images. Since there is no publicly available ground truth for the test images available, we only show the predicted poses in red.

4.3.6.6 Ablation Experiments

We performed additional experiments to show the contributions of our modifications. We evaluate the modifications on the NYU dataset [271], since it has the most accurate annotations, diverse poses, and two different users for evaluation.

Training Data Augmentation In order to evaluate the contribution of the training procedure, we tested the different data augmentation schemes. The results are shown in Table 4.7. Using data augmentation results in an increase in accuracy over 7 mm. Most importantly, augmenting the hand translation accounts for errors in the hand localization method, and augmenting the rotation accounts for rotated hand poses, thus effectively enlarging the training poses.

Although augmenting only the scale does not help as much as augmenting translation or rotation on the NYU dataset, it can help in cases where the size of the users' hand is not accurately determined, *i.e.*, a new user in a practical application. Interestingly,

Augmentation	Average 3D error
No augmentation	19.9 mm
Translation (T)	14.7 mm
Rotation (R)	13.8 mm
Scale (S)	17.1 mm
All (R+T+S)	12.3 mm
All (R+T+S) & no prior aug.	21.7 mm

Table 4.7: Effects of the proposed training procedure on the NYU dataset [271]. By using different data augmentation methods, the accuracy can be significantly increased. In the first row we do not use any data augmentation. In the last row we apply augmentation on the training data, but not for computing the pose prior, showing the importance of having an expressive pose prior.

Localization	Avg. 3D pose error	Loc. 3D error
CoM	13.8 mm	28.1 mm
Refined CoM	12.3 mm	8.6 mm
Ground truth	10.8 mm	0.0 mm

Table 4.8: Impact of hand localization accuracy on NYU dataset [271]. The ground truth localization gives the lowest 3D pose error, but this localization is not applicable in practice. Our refinement of the commonly used center of mass localization (CoM) improves the accuracy by over 1 mm.

computing the prior from augmented 3D hand poses is very important as well. If the data is augmented, but the prior is computed from the original 3D hand poses, the accuracy is worse compared to no data augmentation, since the prior is not expressive enough to capture the variances of the augmented hand poses.

Hand Localization Further, we evaluate the influence of the hand localization on the final 3D joint error. For this experiment, we use the ResNet architecture and all data augmentation. The results are shown in Table 4.8. The highest accuracy can be achieved using the ground truth location of the hand, which is not feasible in practice, since real detectors do not provide perfect hand localization. This indicates, that there is still room for improvement by using a more accurate 3D hand localization method.

Starting with the very simple center of mass localization and by refining the estimated center of mass localization, this step decreases the 3D localization error by almost 20 mm. This in turn improves further the final average 3D pose error by over 1 mm.

Network Architecture We evaluate the impact of the different network architectures in Table 4.9. We use the refined hand localization and all data augmentation for training

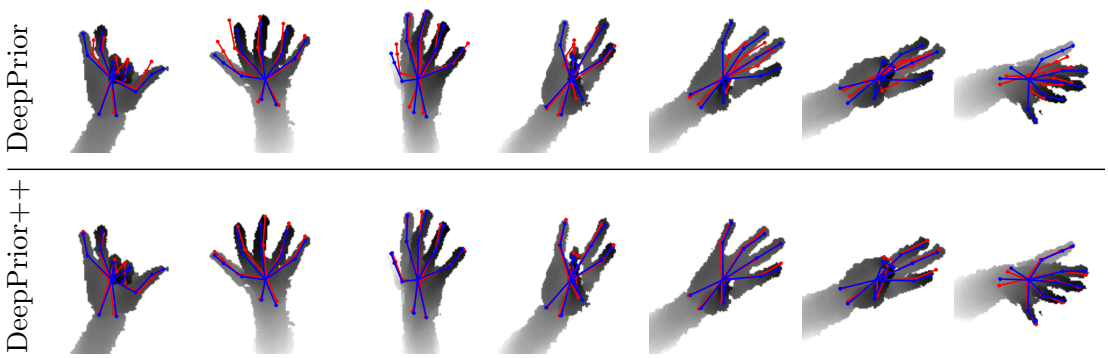


Figure 4.15: Qualitative comparison between DeepPrior and DeepPrior++ on the NYU dataset [271]. We show the inferred 3D joint locations projected on the depth images. Ground truth is shown in blue, the predicted poses in red. The results provided by DeepPrior++ are significantly better than the results from the original DeepPrior, especially on highly articulated poses.

Architecture	Avg. 3D error	*fps*
Original DeepPrior	16.6 mm	100
Original with more filters	13.7 mm	80
ResNet	12.3 mm	30

Table 4.9: Impact of network architecture on the NYU dataset [271]. The more recent ResNet architecture performs significantly better than the original network architecture, even when using the same number of filters as ResNet for the Original architecture (*Original with more filters*). Most importantly, we can still maintain real-time performance with 30 *fps* in our hand tracking application.

the networks. The improved training procedure and better localization already improve the results for the original architecture by more than 3 mm (19.8 mm from DeepPrior). Using the proposed ResNet architecture, the accuracy can be improved by another 4 mm on average, due to the higher capacity of the model. We also evaluated the original architecture, but changed the convolutional layers such that they use the same number of filters as the ResNet architecture, but this architecture is still inferior to the ResNet.

The ResNet architecture is slower than the original implementation, however, it is still able to run at over 30 *fps* on a single GPU, making it applicable to real-time applications. Note that this 30 *fps* is measured in our hand tracking application, which includes camera communication, hand localization, pose estimation, and visualization in a user interface.

4.3.6.7 Qualitative Results

We show several qualitative results in Figure 4.15, where we compare DeepPrior++ to the original DeepPrior. In general, DeepPrior++ provides significantly better results

compared to the original DeepPrior, especially on highly articulated poses. This can be attributed to the data augmentation and better localization, but also to the more powerful *CNN* architecture, which enables the *CNN* to learn highly accurate poses for complex articulations.

4.4 Discussion

We evaluated different network architectures for hand pose estimation by directly regressing the 3D joint locations. We introduced a constrained prior hand model that can significantly improve the joint localization accuracy. Further, we applied a joint-specific refinement stage to increase the localization accuracy. We have shown, that for this refinement a *CNN* with overlapping input patches with different pooling sizes can benefit from both, input resolution and context. We have compared the architectures on two datasets and shown that they achieve very good results both in terms of localization accuracy and speed.

Since the publication of DeepPrior, other works on pose estimation introduced a pose prior in a Deep Learning framework, showing the importance of such prior:

- Several more recent works [73, 74, 322] rely on a bottleneck layer reminiscent to ours in order to improve the accuracy of the predicted 3D hand pose.
- [212] proposed to replace the linear transformation computed by the *PCA* by an encoder. This encoder is trained first as an autoencoder, together with a decoder, to predict a compact representation of the pose. As the decoder has a more complex non-linear form, it brings some improvement in accuracy.
- [263] considers human pose estimation and also uses an autoencoder, but to compute a pose embedding of *larger* dimensions than the original pose, which appears to significantly improves the accuracy in the case of body pose estimation.
- [325] learns a pose prior for estimating the 3D hand joint locations from 2D heatmaps by factorizing the prior into canonical coordinates and a relative motion, while our prior learned with *PCA* does not distinguish between the two.

Maybe a high-level conclusion of the work presented in this chapter is that our community should be careful when comparing approaches: By paying attention to its different steps, we were able to make DeepPrior++ perform significantly better than the original DeepPrior and it performs similarly or better than more recent works, while the key ideas are the same for the two methods.

5

Iterative Hand Pose Estimation with Feedback

Contents

In the previous chapter, we introduced a feed-forward Convolutional Neural Network (CNN) for accurate and fast 3D hand pose estimation. Although achieving high accuracy, it can still make mistakes. We address these mistakes in this chapter and show that we can correct the mistakes made by a *CNN* trained to predict an estimate of the 3D pose by using a feedback loop. The components of this feedback loop are also Deep Networks, optimized using training data. They remove the need for fitting a 3D model to the input data, which requires both a carefully designed fitting function and algorithm. We show that our approach significantly improves the accuracy of the initial estimates and is efficient as our implementation runs at over 400 frames per second (fps) on a single GPU.

5.1 Introduction

A popular approach, as we have introduced in the previous chapter, is to use a discriminative method to predict the position of the joints [122, 183, 254, 256, 271], because they are robust and fast. To refine the pose further, they are often used to initialize an optimization where a 3D model of the hand is fit to the input depth data [12, 50, 187, 197, 201, 227, 241, 279]. Such an optimization remains complex and typically requires the maintaining of multiple hypotheses [186, 187, 201]. It also relies on a criterion to evaluate how well the 3D model fits to the input data, and designing such a criterion is not a simple and straightforward task [12, 50, 241, 259, 266].

In this chapter, we first show how we can get rid of the 3D model of the hand altogether and build instead upon work that learns to generate images from training data [55]. Creating a 3D model of the hand is very difficult since the hand contains numerous muscles, soft tissue, *etc.*, which influence the shape of the hand [239, 251, 259, 266]. We think that our approach could also be applied to other problems where acquiring a 3D model is very difficult.

We then introduce a method that learns to provide updates for improving the current estimate of the pose, given the input depth image and the image generated for this pose estimate as shown in Figure 5.1. By iterating this method a number of times, we can correct the mistakes of an initial estimate provided by a simple discriminative method. All the components are implemented as Deep Networks with simple architectures.

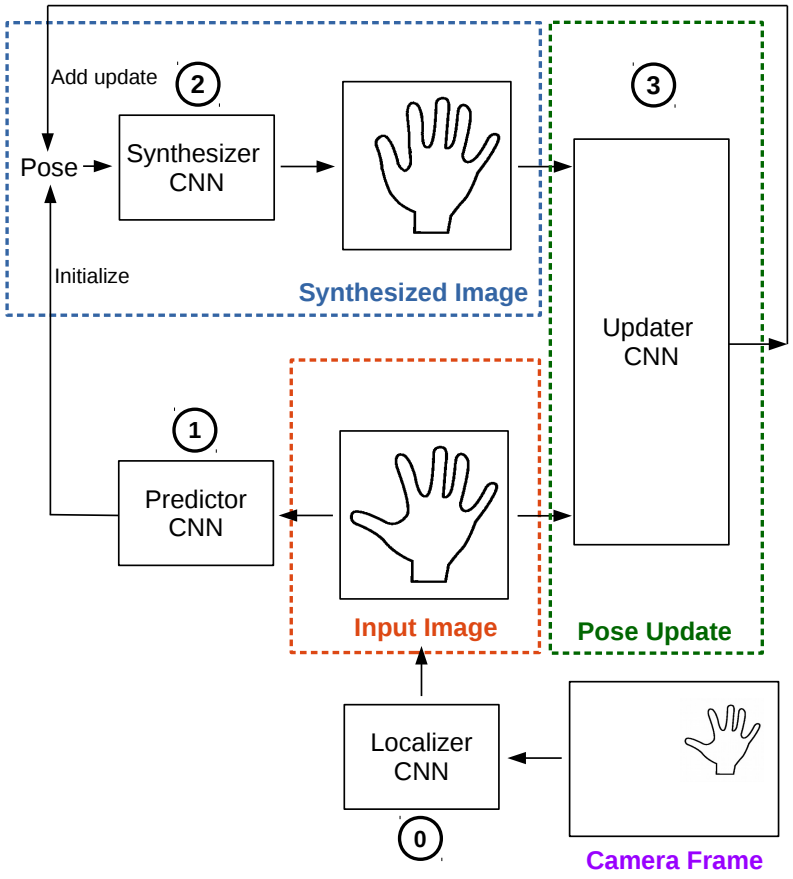


Figure 5.1: Overview of our feedback method. We use a first *CNN* ① to predict an initial estimate of the 3D pose given an input depth image of the hand. The pose is used to synthesize a depth image ②, which is used together with the input depth image to derive a pose update ③. The update is applied to the pose and the process is iterated. Optionally, we can use a localizer *CNN* ⓪ to increase the localization accuracy of the input depth image, as done in Section 4.2.5.

Not only is it interesting to see that all the components needed for hand registration

that used to require careful design can be learned instead, but we will also show that our approach had superior performance when compared to the state-of-the-art methods. It is also efficient; our implementation runs at over 400 *fps* on a single GPU.

Our approach is related to generative approaches [17], in particular [257] which also features a feedback loop reminiscent of ours. However, our approach is deterministic and does not require an optimization based on distribution sampling, on which generative approaches generally rely, but which tends to be inefficient.

In the remainder of the chapter, we first give a short review of related work on feedback methods in Section 5.2. We describe our approach in Section 5.3 and evaluate it in Section 5.4.

5.2 Related Work on Feedback Methods

Our approach presented in this chapter is more related to generative, model-based methods. However, it differs from previous work in the first three building blocks of such conventional generative methods that we presented in Section 2.2.1. We do not use a deformable CAD model of the hand. Instead, we learn from registered depth images to generate realistic depth images of hands, similar to work on inverse graphics networks [55, 132], and other recent work on generating images [76, 133]. This approach is very convenient, since deforming correctly and rendering a CAD model of the hand in a realistic manner requires a significant input of engineering work. Most importantly, it does not require additional training data.

In addition, we do not use a hand-crafted similarity function and an optimization algorithm. We learn rather to predict updates that improve the current estimate of the hand pose from training data, given the input depth image and a generated image for this estimate. Again this approach is very convenient, since it means we do not need to design the similarity function and the optimization algorithm, neither of which is a simple task.

Since we learn to generate images of the hand, our approach is also related to generative approaches, in particular [257], which uses a feedback loop with an updater mechanism akin to ours. It predicts updates for the position from which a patch is cropped from an image, such that the patch fits best to the output of a generative model. However, this step does not predict the full set of parameters. The hidden states of the model are found by a costly sampling process. [175] relies on a given black-box image synthesizer to provide synthetic samples on which the regression network can be trained. It then learns a network to substitute the black-box graphics model, which can ultimately be used to update the pose parameters to generate an image that most resembles the input. In contrast, we learn the generator model directly from training data, without the need for a black-box image synthesizer. Moreover, we will show that the optimization is prone to output infeasible poses or get stuck in local minima and therefore introduce a better approach to improve the pose. The approach of [34] is related to ours, since it also relies on iteratively refining the 2D joint locations for human pose estimation. They use an initial estimate of the 2D

joint locations to generate a heatmap with the joint locations. These heatmaps are stacked to the input image and an update is predicted, which, however, does not handle occlusions well. This process is then iterated a few times. In contrast to our work, their approach only works for 2D images due to limitations of the feedback. Also, their training strategy is different, as it requires a predefined absolute step size and predefined update directions, which can be hard to learn. In contrast, we use latent update directions with a relative step size. [313] introduced Feedback Networks for general purpose iterative prediction. They formulate the feedback as a recurrent operation where a hidden state is passed on to the next iteration. At each iteration, they predict an absolute output and not an update, which is more difficult. Also, their approach does not use any feedback in the input or output space, so their notion of "feedback" is only an internal structure of the predictor. [283] uses an autoencoder to compute an embedding from the 3D hand pose and feed the embedding vector to an image synthesizer that outputs a depth image of the hand. This setup is used to synthesize additional training data for training a pose predictor, but in their setup they only aim at generating synthesized images as close as possible to the real images to train a discriminative predictor.

5.3 Model-based Pose Optimization

In this section, we first give an overview of our method. We then describe in detail the different components of our method: A discriminative approach to predict a first pose estimate, a method able to generate realistic depth images of the hand, and a learning-based method to refine the initial pose estimate using the generated depth images.

5.3.1 Method Overview

Our objective is to estimate the pose $\mathbf{P}$ of a hand in the form of the 3D locations of its joints $\mathbf{P} = \{\mathbf{j}_i\}_{i=1}^{J}$ with $\mathbf{j}_i = (x_i, y_i, z_i)$ from a single depth image $\mathcal{D}$. In practice, $J = 14$ for the dataset we use. We assume that a training set $\mathcal{T} = \{(\mathcal{D}_i, \mathbf{P}_i)\}_{i=1}^{N}$ of depth images labeled with the corresponding 3D joint locations is available.

As explained in the introduction, we first train a predictor *CNN* to predict an initial pose estimate $\widehat{\mathbf{P}}^{(0)}$ in a discriminative manner given an input depth image $\mathcal{D}_{\text{input}}$ centered around the hand location:

$$\widehat{\mathbf{P}}^{(0)} = \text{pred}(\mathcal{D}_{\text{input}}) \ . \tag{5.1}$$

We use a *CNN* to implement the pred(.) function with a standard architecture. More details will be given in Section 5.3.3.

In practice, $\widehat{\mathbf{P}}^{(0)}$ is never perfect, and following the motivation provided in the introduction, we introduce a hand model learned from the training data. As shown in Figure 5.2, this model can synthesize the depth image corresponding to a given pose $\mathbf{P}$, and we will

refer to this model as the *synthesizer CNN*:

$$\mathcal{D}_{\text{synth}} = \text{synth}(\mathbf{P}) \quad . \tag{5.2}$$

We also use a Deep Network to implement the synthesizer.

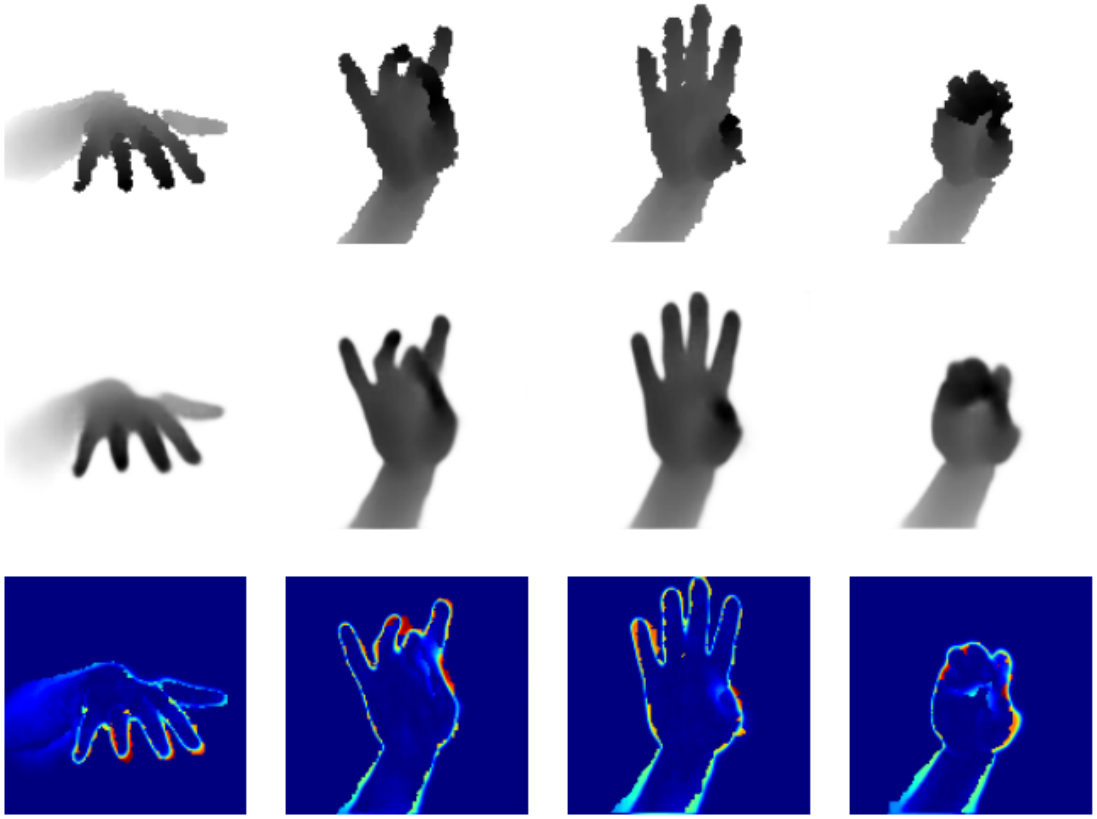
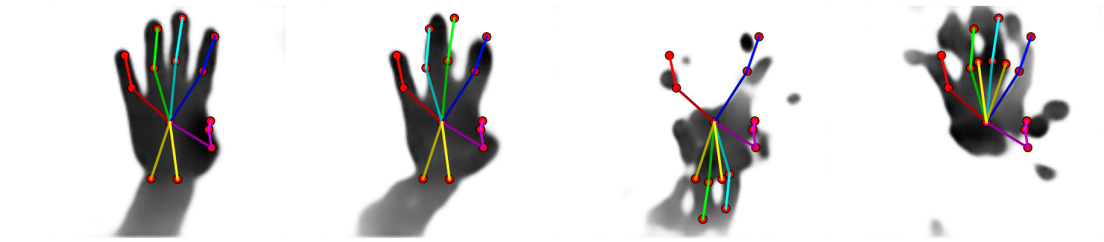


Figure 5.2: Samples generated by the synthesizer *CNN* for different poses from the test set. **Top:** Ground truth depth image. **Middle:** Synthesized depth image using our learned hand model. **Bottom:** Color-coded, pixel-wise difference between the depth images. Red represents large errors, blue represents small errors. The synthesizer *CNN* is able to render convincing depth images for a very large range of poses. The largest errors are located near the occluding contours of the hand, which are noisy in the ground truth images.

A straightforward way of using this synthesizer would consist in estimating the hand pose $\widehat{\mathbf{P}}$ by minimizing the squared loss between the input image and the synthetic one:

$$\widehat{\mathbf{P}} = \arg\min_{\mathbf{P}} \|\mathcal{D}_{\text{input}} - \text{synth}(\mathbf{P})\|^2 \quad . \tag{5.3}$$

This is a non-linear least-squares problem, which can potentially be solved iteratively using $\widehat{\mathbf{P}}^{(0)}$ as initial estimate. However, the objective function of Eqn. (5.3) exhibits many local minima. Moreover, during the optimization of Eqn. (5.3), $\mathbf{P}$ can take values that



(a) **(b)** **(c)** **(d)**

Figure 5.3: Synthesized images for physically impossible poses. Note the colors that indicate different fingers. (a) shows a feasible pose with its synthesized image. (b) shows the synthesized image for the same pose after swapping the ring and middle finger positions. In (c) the ring and middle finger are flipped downwards, and in (d) the wrist joints are flipped upwards.

correspond to physically infeasible poses. For such values, the output synth($\mathbf{P}$) of the synthesizer *CNN* is unpredictable as depicted in Figure 5.3, and this is likely to make the optimization of Eqn. (5.3) diverge or be stuck in a local minimum, as we will show in the experiments in Section 5.4.5.

We therefore introduce a third function that we call the updater(.,.). It learns to predict updates, which are applied to the pose estimate to improve it, given the input image $\mathcal{D}_{\text{input}}$ and the image synth($\mathbf{P}$) produced by the synthesizer *CNN*:

$$\widehat{\mathbf{P}}^{(i+1)} \leftarrow \widehat{\mathbf{P}}^{(i)} + \text{updater}(\mathcal{D}_{\text{input}}, \text{synth}(\widehat{\mathbf{P}}^{(i)})) \quad . \tag{5.4}$$

We iterate this update several times to improve the initial pose estimate. Again, the updater(.,.) function is implemented as a Deep Network.

We detail below how we implement and train the different functions.

5.3.2 Learning the Localizer Function loc(.)

In practice, we require an input depth image centered on the hand location. It is sufficient to rely on a common heuristic that uses the center of mass for localizing the hand [183, 254]. However, we can optionally further improve this heuristic by introducing the localizer *CNN* loc(.), as we already used in Section 4.2.5. We still use the center of mass for the initial localization from the depth camera frame, but apply an additional refinement step that improves the final accuracy [173, 181]. We refer to this optional step as *improved feedback* in our experiments in Section 5.4.4. This refinement step relies on the localizer *CNN*. The *CNN* is applied to the 3D bounding box centered on the center of mass, and is trained to predict the location of the metacarpophalangeal (MCP) joint of the middle finger, which we use as referential. The localizer *CNN* has a simple network architecture, which is shown in Figure 4.4.

We train the localizer *CNN*, parametrized by ϕ, by optimizing the cost function:

$$\widehat{\phi} = \arg\min_{\phi} \sum_{(\mathcal{D},\mathbf{l})\in\mathcal{T}} \|\mathrm{loc}_{\phi}(\mathcal{D}) - \mathbf{l}\|^2 \ . \tag{5.5}$$

$\mathbf{l}$ denotes the offset in image coordinates and depth between the center of mass and the MCP of the hand. For inference, we crop a depth image centered on the center of mass from the depth camera frame, then predict the MCP location by applying loc(.) to this crop, and finally crop again from the depth camera frame using the predicted location.

We crop the training depth images and the original input depth images around the locations provided by $\mathrm{loc}_{\phi}(.)$ for these images: $\mathcal{D}$ in Eqn. (5.6), (5.7), (5.8), (5.10), (5.11), (5.12) therefore denotes a training depth image after cropping, and $\mathcal{D}_{\text{input}}$ the original input depth image after cropping.

5.3.3 Learning the Predictor Function pred(.)

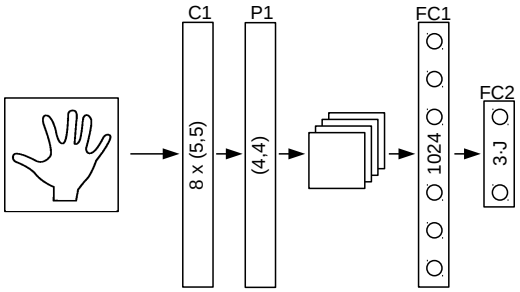


Figure 5.4: Network architecture of the predictor *CNN* used for initial pose prediction. C denotes a convolutional layer with the number of filters and the filter size inscribed, FC a fully connected layer with the number of neurons, and P a max-pooling layer with the window size.

The predictor *CNN* is implemented as a vanilla neural network. The network consists of a convolutional layer with 5×5 filter kernels producing 8 feature maps. These feature maps are max-pooled with 4×4 windows, followed by a hidden layer with 1024 neurons and an output layer with one neuron for each joint and dimension, *i.e.*, $3 \cdot J$ neurons. The predictor is parametrized by Φ, which is obtained by minimizing:

$$\widehat{\Phi} = \arg\min_{\Phi} \sum_{(\mathcal{D},\mathbf{P})\in\mathcal{T}} \|\mathrm{pred}_{\Phi}(\mathcal{D}) - \mathbf{P}\|^2 + \gamma\|\Phi\|_2^2 \ , \tag{5.6}$$

where the second term is a regularizer for weight decay with $\gamma = 0.001$.

5.3.4 Learning the Synthesizer Function synth(.)

We also use a *CNN* to implement the synthesizer *CNN* synth(.), and we train it using the set $\mathcal{T}$ of annotated training pairs. The network architecture is strongly inspired by [55], and is shown in Figure 5.6. It consists of four hidden layers, which learn an initial latent representation of feature maps apparent after the fourth fully connected layer FC4. These latent feature maps are followed by several unpooling and convolution layers. The unpooling operation, used for example by [55, 76, 314], is the inverse of the max-pooling operation: The feature map is expanded, in our case by a factor of two along each image dimension as shown in Figure 5.5. The emerging "holes" are filled with zeros. The expanded feature maps are then convolved with trained 3D filters to generate another set of feature maps. These unpooling and convolution operations are applied subsequently. The last convolution layer combines all feature maps to generate a depth image.

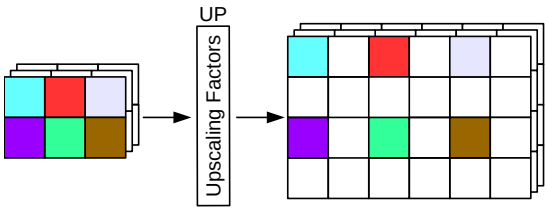


Figure 5.5: Unpooling layer with an upscaling factor of two. This layer takes dense feature maps as input and expands them by the upscaling factor along each direction. This expansion creates "holes" that are filled with zeros.

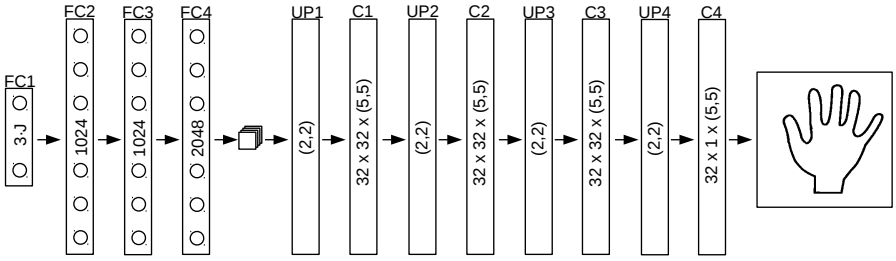


Figure 5.6: Network architecture of the synthesizer *CNN* used to generate depth images of hands given their poses. The input of the network is the hand pose. The fully connected hidden layers create a 2048-dimensional latent representation at FC4 which is reshaped into 32 feature maps of size 8×8. The feature maps are gradually enlarged by successive unpooling and convolution operations. The last convolution layer combines the feature maps to derive a single depth image of size 128×128. All layers have rectified-linear units, except the last layer which has tanh units. C denotes a convolutional layer with the number of filters and the filter size inscribed, FC a fully connected layer with the number of neurons, and UP an unpooling layer with the upscaling factor.

We learn the parameters $\widehat{\Theta}$ of the network by minimizing the difference between the generated depth images synth($\mathbf{P}$) and the training depth images $\mathcal{D}$ as:

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{(\mathcal{D},\mathbf{P})\in\mathcal{T}} \frac{1}{|\mathcal{D}|} \|\text{synth}_{\Theta}(\mathbf{P}) - \mathcal{D}\|^2 \ . \tag{5.7}$$

We perform the optimization in a layer-wise fashion. We start by training the first 8×8 feature map. Then we gradually extend the output size by adding another unpooling and convolutional layer and train again, which achieves lesser errors than end-to-end training in our experience.

The synthesizer *CNN* is able to generate accurate images, maybe surprisingly well for such a simple architecture. The median pixel error on the test set is only 0.1 mm. However, the average pixel error is 8.9 ± 28.5 mm. This is mostly due to noise in the input images along the outline of the hand, which is smoothed away by the synthesizer *CNN*. The average depth accuracy of the sensor is ± 1 mm [200].

5.3.5 Learning the Updater Function updater(.,.)

The updater *CNN* updater(.,.) takes two depth images as input. As already stated in Eqn. (5.4), at run-time, the first image is the input depth image, the second image is the image returned by the synthesizer for the current pose estimate. Its output is an update that improves the pose estimate. The architecture, shown in Fig. 5.7, is inspired by the Siamese network [42]. It consists of two identical paths with shared weights. One path is fed with the observed image and the second path is fed with the image from the synthesizer. Each path consists of four convolutional layers. We do not use max-pooling here, but a filter stride [113, 151] to reduce the resolution of the feature maps. We experienced inferior accuracy with max-pooling compared to that with stride, probably because max-pooling introduces spatial invariance [222] that is not desired for this task. The feature maps of the two paths are concatenated and fed into a fully connected network that outputs the update.

Ideally, the output of the updater *CNN* should bring the pose estimate to the correct pose in a single step as shown in Figure 5.8a. This is a very difficult problem, though, and we could not get the network to reduce the initial training error within a reasonable timeframe. However, our only requirement from the updater *CNN* is for it to output an update that brings us closer to the ground truth as shown in Figure 5.8b. We iterate this update procedure to get closer step-by-step. Thus, the update should follow the inequality

$$\|\mathbf{P} + \text{updater}(\mathcal{D}, \text{synth}(\mathbf{P})) - \mathbf{P}_{\text{GT}}\| < \lambda \|\mathbf{P} - \mathbf{P}_{\text{GT}}\| \ , \tag{5.8}$$

where $\mathbf{P}_{\text{GT}}$ is the ground truth pose for image $\mathcal{D}$, and $\lambda \in [0, 1]$ is a multiplicative factor that specifies a minimal improvement. We use $\lambda = 0.6$ in our experiments.

We optimize the parameters Ω of the updater *CNN* by minimizing the following cost

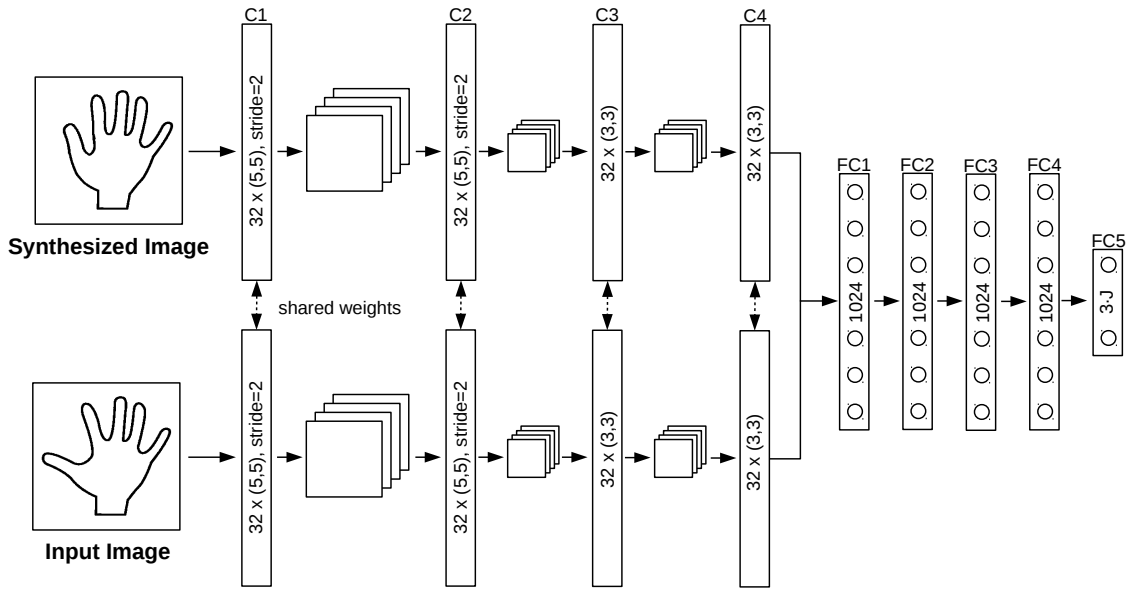


Figure 5.7: Architecture of the updater *CNN*. The network contains several convolutional layers that use a filter stride to reduce the size of the feature maps. The final feature maps are fed into a fully connected network. All layers have rectified-linear units, except the last layer has linear units. The pose update is used to refine the initial pose and the refined pose is again fed into the synthesizer *CNN* to iterate the whole procedure. As in Figure 5.6, C denotes a convolutional layer, and FC a fully connected layer.

function

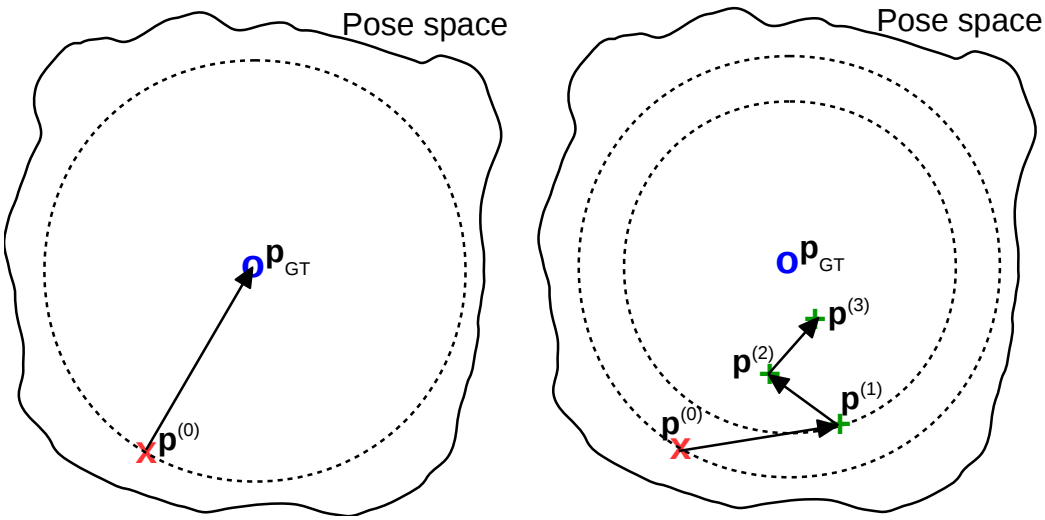
$$\mathcal{L} = \sum_{(\mathcal{D},\mathbf{P})\in\mathcal{T}} \sum_{\mathbf{P}'\in\mathcal{T}_\mathcal{D}} \max(0, \|\mathbf{P}'' - \mathbf{P}\| - \lambda\|\mathbf{P}' - \mathbf{P}\|) \ , \tag{5.9}$$

where $\mathbf{P}'' = \mathbf{P}' + \text{updater}_\Omega(\mathcal{D}, \text{synth}(\mathbf{P}'))$, and $\mathcal{T}_\mathcal{D}$ is a set of poses. The introduction of the synthesizer *CNN* allows us to virtually augment the training data and add arbitrary poses to $\mathcal{T}_\mathcal{D}$, which the updater *CNN* is then trained to correct.

The set $\mathcal{T}_\mathcal{D}$ contains the ground truth poses $\mathbf{P}$, for which the updater *CNN* should output a zero update. We further add as many meaningful deviations from that ground truth as possible, which the updater *CNN* might perceive during testing and be asked to correct. We start by including the output pose of the predictor *CNN* $\text{pred}(\mathcal{D})$, which is used during testing as initialization of the update loop. Additionally, we add copies with small Gaussian noise for all poses. This creates convergence basins around the ground truth, in which the predicted updates point towards the ground truth, as we show in the evaluation, and helps to explore the pose space.

After every two epochs, we augment the set by applying the current updater *CNN* to the poses in $\mathcal{T}_\mathcal{D}$, that is, we add the set

$$\{\mathbf{P}_2 \mid \exists\mathbf{P} \in \mathcal{T}_\mathcal{D} \text{ s.t. } \mathbf{P}_2 = \mathbf{P} + \text{updater}(\mathcal{D}, \text{synth}(\mathbf{P}))\} \tag{5.10}$$



(a) Direct prediction. **(b)** Iterative prediction.

Figure 5.8: Ideally, we would like a predictor to update an initial pose (×) to the ground truth pose (○) in a single step, as schematized in 2D in (a). However, this is very challenging in practice. Our iterative pose optimization in high-dimensional space is shown in (b). We start at an initial pose (×) and want to converge to the ground truth pose (○) that maximizes image similarity. Our updater *CNN* generates updates for each pose (+) that bring us closer to the ground truth pose. The updates are predicted from the synthesized image of the current pose estimate and the observed depth image.

to $\mathcal{T}_{\mathcal{D}}$. This forces the updater *CNN* to learn to further improve on its own outputs.

In addition, we sample from the current distribution of errors across all the samples, and add these errors to the poses, thus explicitly focusing the training on common deviations. This is different from the Gaussian noise and helps to predict correct updates for larger initialization errors.

We show a more intuitive interpretation of the training in Figure 5.9. We plot an error function E over the difference between a pose and the ground truth pose. A typical metric for this function is for example the squared difference between the input depth image and the synthesized image. This function can exhibit local minima that makes optimization of the appearance in image space difficult. However, our method is more robust to such local minima. The red dot shows the ground truth pose. In a first step, we train the updater *CNN* to get closer to the ground truth by adding Gaussian noise to the ground truth location. We do this several times. Further, we use the predicted pose of the predictor *CNN* on the training set as starting point and predict an update for this pose. Here, the pose is located in a local minima when only considering the difference function between the images. However, we can escape such a minima by improving on the pose without any assumptions on the similarity function E. Thus, the actual latent error function that we learn might not exhibit a local minima there and we can improve the

pose. Again, we sample additional poses around the predictions by adding small Gaussian noise. In addition, we add previously predicted updates and sample additional poses around the predicted updates. Last, we add further random samples from the current error distribution of the updater to explore the pose space further.

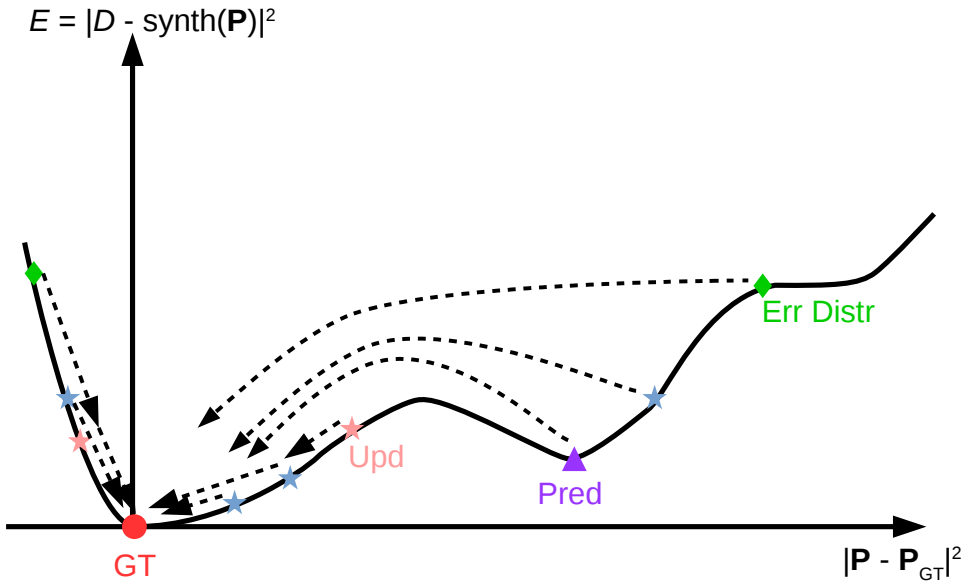


Figure 5.9: Intuitive interpretation of the training. The y-axis denotes the squared difference in image space between the input image and the synthesized image and the x-axis the difference between estimated pose and ground truth pose in pose space. Hence, this plot depicts the image difference as a function of the pose difference. The red dot depicts the ground truth pose, green diamonds depict samples from the error distribution of the predictor, the pink triangle depicts the initial prediction, blue stars depict random poses sampled around the ground truth and the prediction, and salmon stars depict predicted updates from previous iterations. Ideally, the global minimum of this function should be at the location where the estimated pose is equal to ground truth pose. However, there can be many local minima in the difference, *e.g.*, due to depth noise or self-similarities of the hand. We train the updater *CNN* to predict updates (shown as dashed lines) that all bring the pose closer to the ground truth pose, and can escape local minima of the image difference function.

5.3.6 Learning all Functions Jointly

So far, we have considered optimizing all functions separately, *i.e.*, first training the synthesizer *CNN* and the predictor *CNN*, and then using them to train the updater *CNN*. However, it is theoretically possible to train the three networks together. One iteration can be expressed in terms of the current pose $\mathbf{P}$ by introducing the following iter(.) function:

$$\text{iter}(\mathbf{P}) = \mathbf{P} + \text{updater}(\text{synth}(\mathbf{P}), \mathcal{D}) \quad . \tag{5.11}$$

The pose estimate $\widehat{\mathbf{P}}^{(n)}$ after n iterations can be written as:

$$\widehat{\mathbf{P}}^{(n)} = \text{iter}(\ldots \text{iter}(\text{pred}(\mathcal{D}))\ldots) \ . \tag{5.12}$$

pred(.), synth(.), and updater(.,.) can now be trained by minimizing:

$$\{\widehat{\Phi}, \widehat{\Theta}, \widehat{\Omega}\} = \underset{\{\Phi,\Theta,\Omega\}}{\arg\min} \sum_{(\mathcal{D},\mathbf{P})\in\mathcal{T}} \|\widehat{\mathbf{P}}^{(n)} - \mathbf{P}\|^2 \ . \tag{5.13}$$

The function iter(.) can be seen as a Recurrent Neural Network (RNN) that depends on the input depth image $\mathcal{D}$. In comparison to RNNs, our method makes training simpler, intermediary steps easier to understand, and the design of the networks' architectures easier.

We tried to optimize Eqn. (5.13) starting from a random network initialization, but the optimization did not converge to a satisfying solution. Also, when using the pretrained synthesizer *CNN* and predictor *CNN* as initialization, the optimization converges, but this leads to similar accuracy, indicating that end-to-end optimization is possible, but not useful here. Moreover, the intermediate results, *i.e.*, the synthesized depth images, are less interpretable as the hand is barely recognizable. Splitting the optimization problem as we do makes therefore the optimization easier, at no loss of accuracy.

5.4 Evaluation

In this section we evaluate our proposed method on the NYU dataset [271], a challenging real-world benchmark for hand pose estimation. First, we describe how we train the networks. Then, we introduce the training dataset. Furthermore, we evaluate our method qualitatively and quantitatively.

5.4.1 Network Optimization

We optimize the network parameters using gradient descent, specifically using the ADAM method [125] with default hyper-parameters. The batch size is 64. The learning rate decays over the epochs and starts with 0.001. The networks are trained for 100 epochs.

5.4.2 Training Dataset

We evaluated our method on the NYU dataset [271]. For the image data we use the same preprocessing step as described in Chapter 4. We also considered other datasets for this task; unfortunately, no further dataset was suitable. The ICVL dataset of Tang *et al.* [254] has large annotation errors, that cause blurry outlines and "ghost" fingers in the synthesized images as shown in Figure 5.10, which are not suitable for our method. The datasets of [201, 241, 299] provide too little training data to train meaningful models.

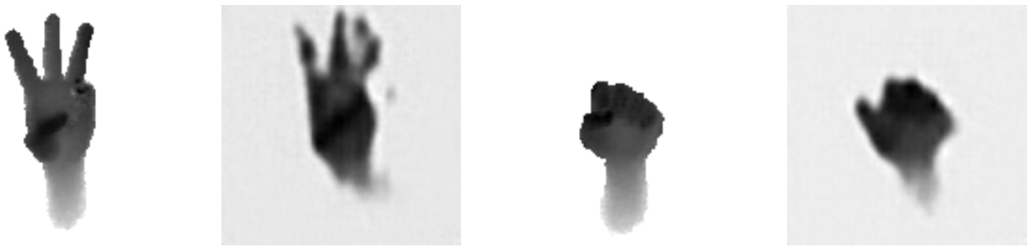
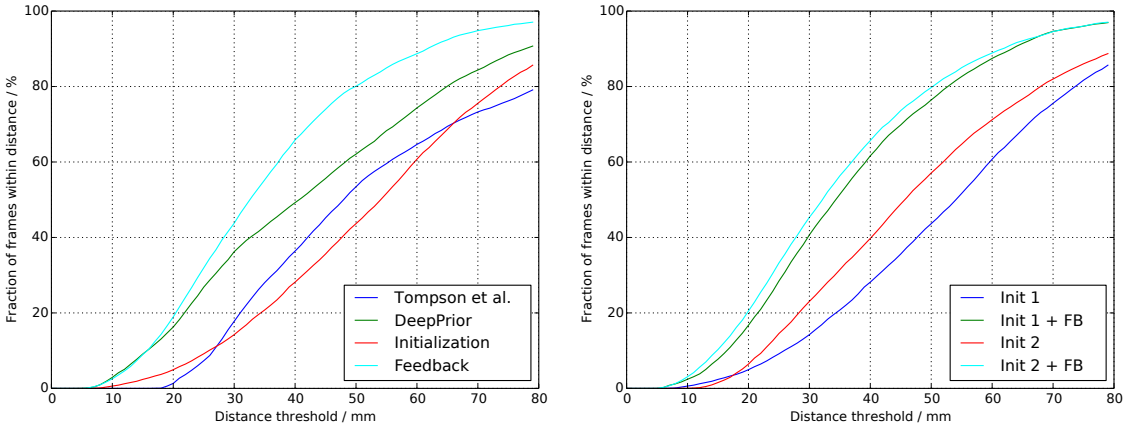


Figure 5.10: The effects of erroneous annotations in the training data of [254]. The figures show input images together with synthesized images which exhibit very blurry outlines and "ghost" fingers. The synthesizer learns the erroneous annotations and interpolates between inconsistent ones, causing such artifacts that limit the applicability in our method.

5.4.3 Comparison with Baseline



(a) **(b)**

Figure 5.11: Quantitative evaluation of pose optimization. Both figures show the fraction of frames where all joints are within a maximum distance. A higher area under the curve denotes better results. In (a) we compare our method to the baseline of Tompson *et al.* [271] and DeepPrior introduced in Chapter 4. Although our initialization is worse than both baselines, we can boost the accuracy of the joint locations using our proposed feedback method. In (b) we compare different initializations. *Init 1* is the simple predictor presented in Section 5.3.3. We used our more sophisticated DeepPrior that we introduced in Chapter 4 for a more accurate initialization, denoted as *Init 2*. *+FB* indicates the application of our feedback method. The better initialization helps obtain slightly more accurate results, however, our much simpler and faster predictor is already sufficient for our method as initialization.

We show the benefit of using our proposed feedback loop to increase the accuracy of the 3D joint localization. For this, we compare to Tompson *et al.* [271], and to DeepPrior. For [271], we augment their 2D joint locations with the depth from the depth images, as

done in [271]. In case this estimate is outside the hand cube we assign the ground truth depth, thus favorably mitigating large errors in those cases.

The quantitative comparison is shown in Figure 5.11a. It denotes the fraction of test samples that have all predicted joints below a given maximum Euclidean distance from the ground truth, which is generally regarded as being very challenging. Thus a single erroneous joint results in the deterioration of the whole hand pose.

While the baseline of [271] and [183] have an average Euclidean joint error of 21 mm and 20 mm respectively, our proposed method reaches an error reduction to 16.5 mm on the dataset. The initialization with the simple and efficient proposed predictor has an error of 27 mm. We further show an evaluation of different initializations in Figure 5.11b. When we use a more complex initialization [183] with an error of 23 mm, we can decrease the average error to 16 mm.

5.4.4 Improving the Results

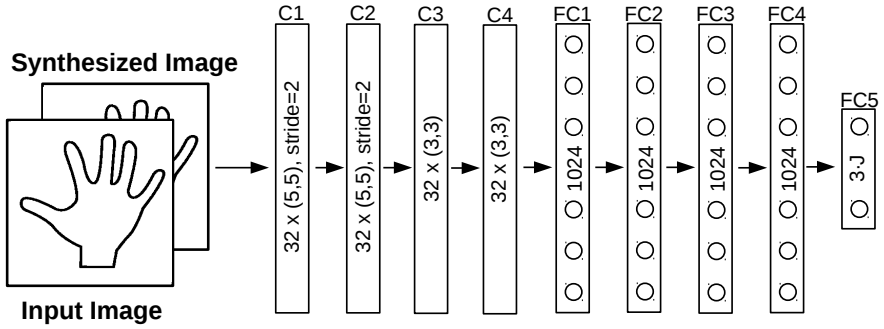


Figure 5.12: Architecture of the updater *CNN*. The input to the network is the input image stacked to the synthesized image. The network contains several convolutional layers that use a filter stride to reduce the size of the feature maps. The final feature maps are fed into a fully connected network. All layers have rectified-linear units, except the last layer has linear units. The pose update is used to refine the initial pose and the refined pose is again fed into the synthesizer *CNN* to iterate the whole procedure. As in Figure 5.6, C denotes a convolutional layer, and FC a fully connected layer.

We further extend the our approach and implement recent advances as described in Section 4.2.5. Specifically, we use a refinement for the hand localization, and further augment the training data during training. Also, we slightly adjust the network architecture as shown in Figure 5.12. Instead of using two separate streams and concatenating the feature maps as shown in Figure 5.7, we concatenate the input image with the synthesized image and use this two-channel image as input to the updater *CNN*. The intuition behind this is that the feature extraction of the two stream network introduces spatial invariance due to the filter stride and by directly concatenating the input depth images, the network can directly learn features that correspond to the differences in the images.

Method	Average 3D error
Neverova *et al.* [178]	14.9 mm
Crossing Nets [283]	15.5 mm
Lie-X [300]	14.5 mm
REN [82]	13.4 mm
DeepPrior++ (Section 4.2.5)	12.3 mm
Feedback (Section 5.3)	16.2 mm
Hand3D [52]	17.6 mm
DISCO [22]	20.7 mm
DeepModel [321]	16.9 mm
Pose-REN [39]	11.8 mm
3DCNN [73]	10.6 mm
Improved Feedback	10.8 mm

Table 5.1: Quantitative evaluation on the NYU dataset [271]. We compare the average Euclidean 3D error of the predicted poses with state-of-the-art methods on the NYU dataset.

We show the benefits of adding the described improvements. For this, we compare our method to recent state-of-the-art methods: *DeepPrior++* introduced in Section 4.2.5 that integrates a prior on the 3D hand poses into a Deep Network; *REN* [82] relies on an ensemble of Deep Networks, each operating on a region of the input image; *Lie-X* [300] uses a sophisticated tracking algorithm constrained to the Lie group; *Crossing Nets* [283] uses an adversarial training architecture; Neverova *et al.* [178] proposed a semi-supervised approach that incorporates a semantic segmentation of the hand; *DeepModel* [321] integrates a 3D hand model into a Deep Network; *DISCO* [22] learns the posterior distribution of hand poses; *Hand3D* [52] uses a volumetric CNN to process a point cloud, similar to *3DCNN* [73].

We show quantitative comparisons in Table 5.1, which compares the different methods we consider using the average 3D error. For our feedback loop, we use DeepPrior++ for initialization as described in Section 4.2.5, which performs already very accurately. Still, our feedback loop can reduce the error from 12.2 mm to 10.8 mm.

5.4.5 Image-Based Hand Pose Optimization

We mentioned in Section 5.3.1 that the attempt may be made to estimate the pose by directly optimizing the squared loss between the input image and the synthetic one as given in Eqn. (5.3) and we argued that this does not in fact work well. We now demonstrate this empirically.

We used the powerful L-BFGS-B algorithm [30], which is a box constrained optimizer, to solve Eqn. (5.3). We set the constraints on the pose in such a manner that each joint coordinate stays inside the hand cube.

The minimizer of Eqn. (5.3), however, does not correspond to a better pose in general,

as shown in Figure 5.13. Although the generated image looks very similar to the input image, the pose does not improve, moreover it even often becomes worse. Several reasons can account for this. The depth input image typically exhibits noise along the contours, as in the example of Figure 5.13. After several iterations of L-BFGS-B, the optimization may start corrupting the pose estimate with the result that the synthesizer generates artifacts that fit the noise. As shown in Figure 5.14, the pose after optimization is actually worse than the initial pose of the predictor.

Furthermore, the optimization is prone to local minima due to a noisy error surface [201]. We also tried Particle Swarm Optimization (PSO) [186, 201, 227] a genetic algorithm popular for hand pose optimization, and obtained similar results. This tends to confirm that the bad performance comes from the objective function of Eqn. (5.3) rather than the optimization algorithm.

By contrast, in Figure 5.15 we show the predicted updates for different initializations around the ground truth joint location with our updater. It predicts updates that move the pose closer to the ground truth, for almost all initializations.

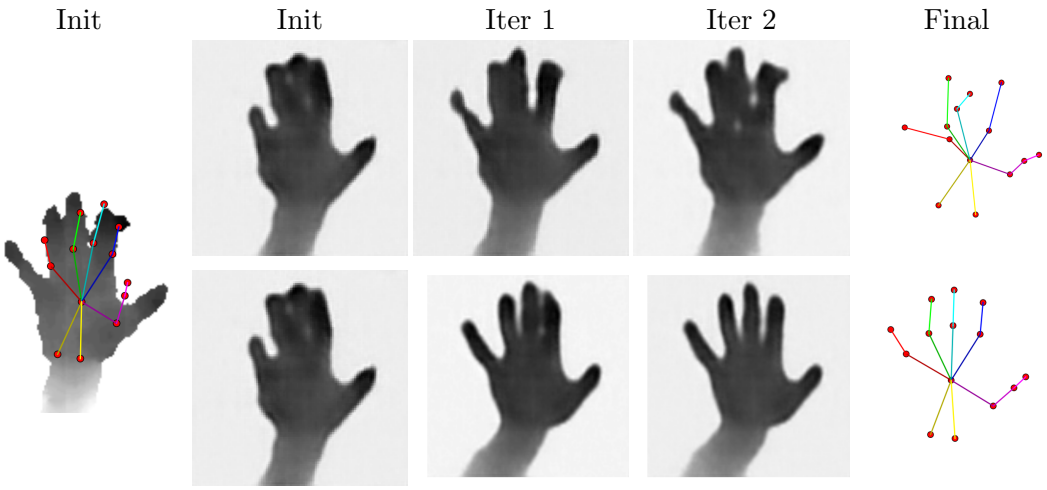


Figure 5.13: Comparison with image-based pose optimization. **(Top)** results for image-based optimization, and **(bottom)** for our proposed method. From left to right: input depth image with initial pose, synthesized image for initial pose, after first, second iteration, and final pose. Minimizing the difference between the synthesized and the input image does not induce better poses. Thanks to the updater, our method can fit a good estimate.

5.4.6 Analysis of Updater CNN

For a more detailed interpretation of the updater *CNN*, let us assume that the updater *CNN* learns the gradient $\frac{\partial E(\mathcal{D},\mathbf{P})}{\partial \mathbf{P}} = \text{updater}(\mathcal{D}, \text{synth}(\mathbf{P}))$ of a latent error function

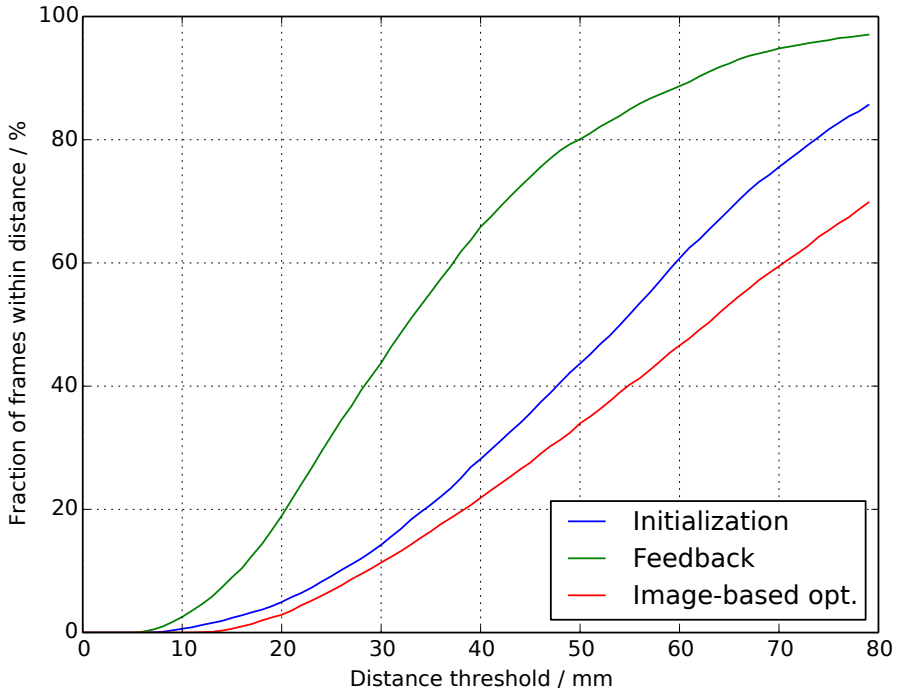


Figure 5.14: Comparing image-based optimization and our feedback loop. A higher area under the curve denotes better results. The image-based optimization actually tends to result in a deterioration of the initial pose estimate. Our proposed optimization method performs significantly better.

$E(\mathcal{D}, \mathbf{P})$. We can perform integration to obtain the function:

$$E(\mathcal{D}, \mathbf{P}) = \int \text{updater}(\mathcal{D}, \text{synth}(\mathbf{P}))\, \mathrm{d}\mathbf{P} \ . \tag{5.14}$$

However, we cannot easily visualize the resulting function $E(\mathcal{D}, \mathbf{P})$ in a meaningful way due to the high dimensionality of the pose space. We therefore simplify the integration for visualization purposes and integrate over two coordinates x, y around a fixed pose $\mathbf{P}$:

$$F(x, y) = \int_{x,y} \text{updater}(\mathcal{D}, \text{synth}(\mathbf{P} + [x, y, 0]))\, \mathrm{d}x\mathrm{d}y \ . \tag{5.15}$$

We numerically solve this integral on a discretized grid of (x, y) locations around the ground truth pose and visualize the result in Figure 5.16. One can clearly see the effect of training the updater, which leads to a convex function with a minimum at the ground truth location. We repeated this visualization for different test samples with different $\mathbf{P}$ and $\mathcal{D}$ and observed similar shaped functions.

The previous visualization only works for a single pose. However, we can visualize

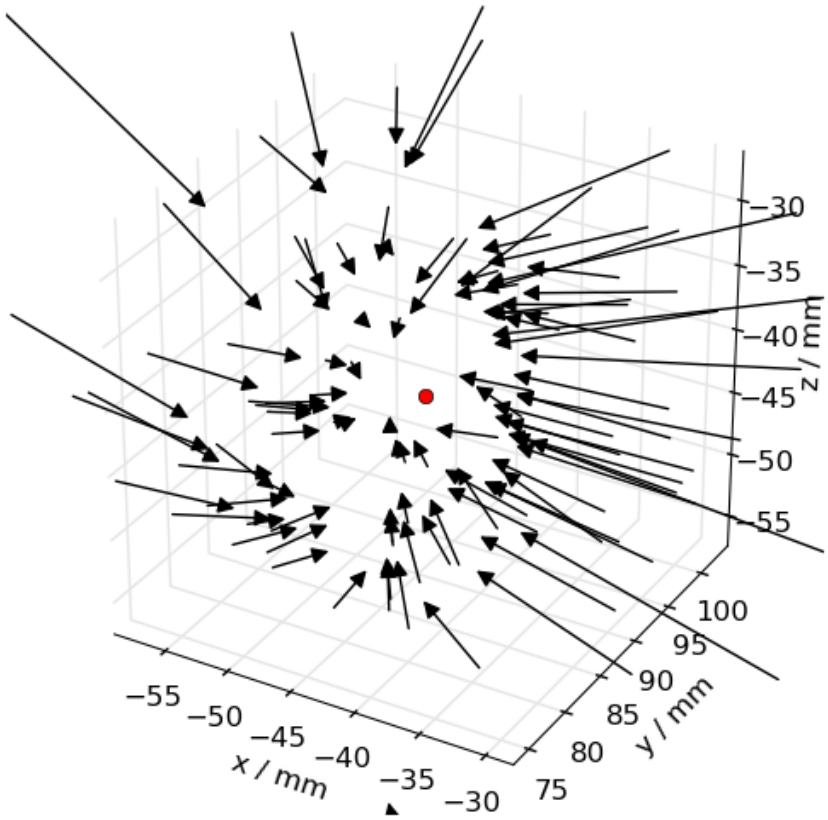


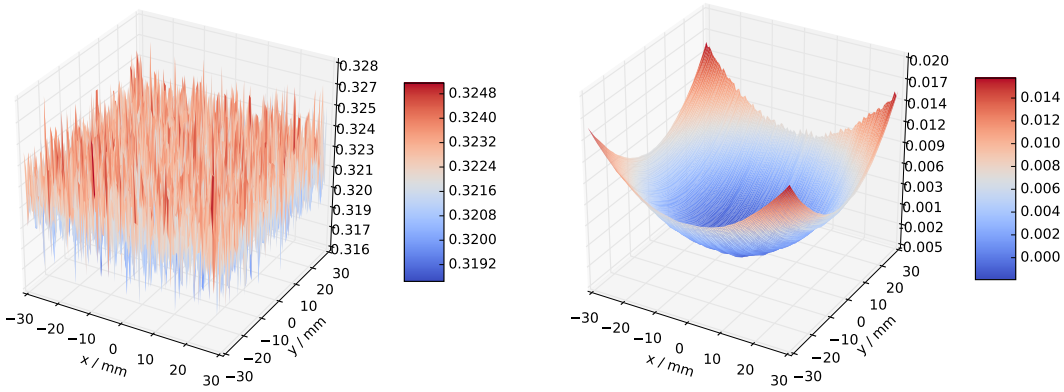
Figure 5.15: Predicted updates around the ground truth joint position, denoted as ●. We initialize noisy joint locations around the ground truth location and show the pose updates as vectors predicted by our updater *CNN*. The start and end of the vectors denote the initial and the updated 3D joint location. All the different updates bring us closer to the ground truth joint location.

multiple slices though the function as an integral along a line. By using an integration along a line trough the ground truth pose connecting two poses $\mathbf{P}_0$ and $\mathbf{P}_1$, we can average the lines between different poses to get a sense of the effect of different starting poses. Thus, we numerically calculate:

$$F(t) = \int_{\mathbf{P}_0}^{\mathbf{P}_1} \text{updater}(\mathcal{D}, \text{synth}(\mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t/N))\,\mathrm{d}t\ , \tag{5.16}$$

where we use $N = 100$ samples along the line. We show this visualization in Figure 5.17. For comparison, we also plot the Euclidean distance between the depth image $\mathcal{D}$ and the synthesized image synth(.):

$$G(t) = \|\mathcal{D} - \text{synth}(\mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t/N)\|^2\ . \tag{5.17}$$



(a) Error function without training. **(b)** Error function after training.

Figure 5.16: Visualization of the latent error function before and after training. Without training, the predictions of the updater *CNN* do not make any sense and thus result in a very noisy function. However, after training the function shows a minimum at the ground truth location, which is a desired property of an error function.

The visualization shows that the latent function $F(t)$ of the updater *CNN* has a distinctive minimum located at the ground truth pose, whereas the difference $G(t)$ between the depth image and the synthesized image does not show such a distinctive minimum. Looking at the variance of the different curves reveals that this minimum is consistent for different poses for the updater *CNN* and this is not the case for the image difference $G(t)$.

5.4.7 Joint-Specific Evaluation

We show the evaluation of the average error for each joint in Figure 5.18. A very small error can be observed for the hand crop location C, since aligning the global silhouette works especially well with our feedback method.

5.4.8 Number of Training Samples

We further evaluate the influence of the number of training samples on the accuracy. We plot the average 3D error over the fraction of training data used for training in Figure 5.19. For this experiment we randomly subsample the training data. The accuracy is only slightly worse when using 50% of the training data, but then significantly decreases when using less training data. Although our method for training the updater *CNN* described in Section 5.3.5 can explore a large pose space, the image quality of the synthesizer *CNN* deteriorates when using less training samples, which makes the updates less effective.

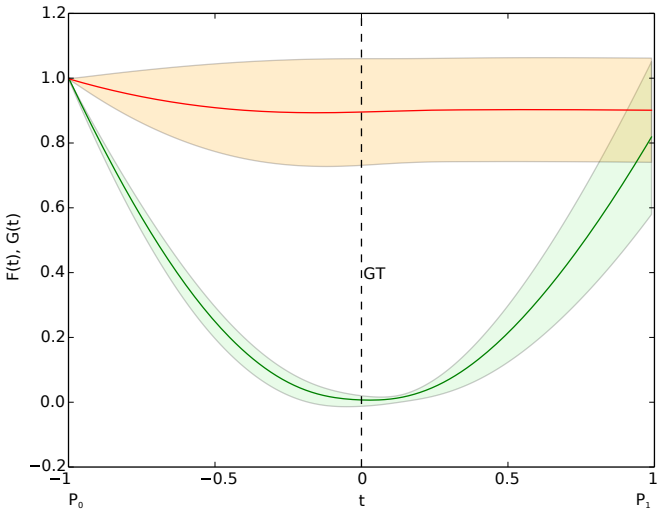


Figure 5.17: Visualization of the latent error function. We show slices through the obtained latent error function of our updater *CNN* in green and the Euclidean distance between the input image and the synthesized image in orange. We plot the average with the variance over 200 randomly sampled slices. The error function of our updater *CNN* has a minimum at the location of the ground truth pose, whereas the difference between the images does not show such a desired property.

5.4.9 Qualitative Results

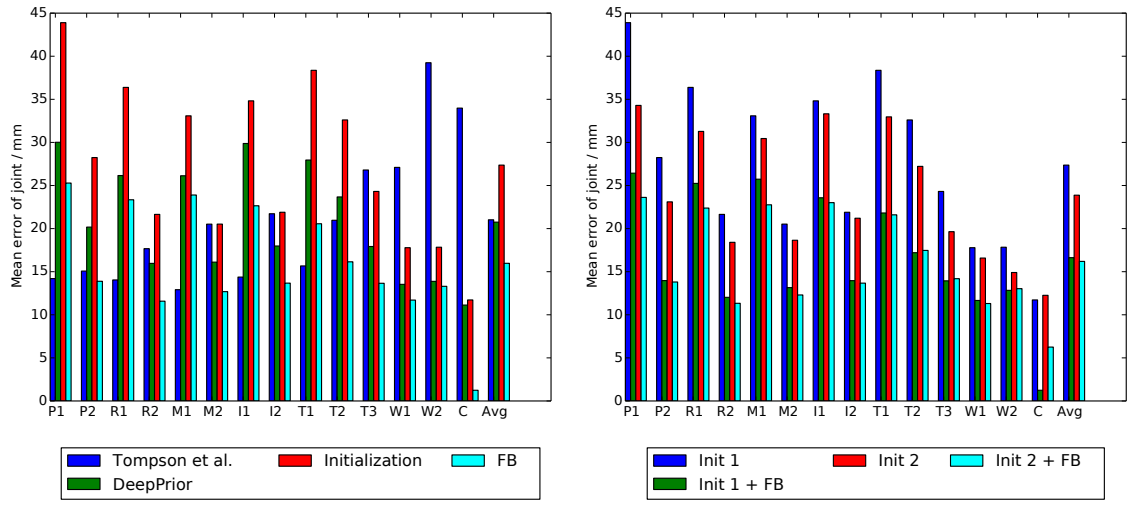
Figure 5.20 shows some qualitative examples. For some examples, the predictor provides already a good pose, which we can still improve, especially for the thumb. For worse initializations, also larger updates on the pose can be achieved by our proposed method, to better explain the evidence in the image.

5.4.10 Runtime

Our method is implemented in Python using the Theano library [15] and we run the experiments on a computer equipped with an Intel Core i7, 16GB of RAM, and an nVidia GeForce GTX 780 Ti GPU. Training takes about ten hours for each *CNN*.

The runtime is composed of the discriminative initialization that takes 0.07 ms, the updater network takes 1.2 ms for each iteration, and that already includes the synthesizer with 0.8 ms. In practice we iterate our updater twice, thus our method performs very fast at over 400 *fps* on a single GPU. The runtime of our method compares favorably with other model-based methods ranging between 12 and 60 *fps* [165, 186, 201, 241, 299].

For the improved feedback in Section 5.4.4, we use the more complex DeepPrior++ as predictor *CNN* for the initial poses, which takes 20 ms. Still, our method runs at over 40 *fps*.



(a) **(b)**

Figure 5.18: Quantitative evaluation of pose optimization. Both figures show the average joint error. In (a) we compare our method to the baseline of Tompson *et al.* [271] and DeepPrior introduced in Chapter 4. Although our initialization is worse than both baselines, we can boost the accuracy of the joint locations using our proposed feedback method *FB*. In (b) we compare different initializations. *Init 1* is the simple predictor presented in Section 5.3.3. We used our more sophisticated DeepPrior that we introduced in Chapter 4 for a more accurate initialization, denoted as *Init 2*. *+FB* indicates the application of our feedback method. The better initialization helps to get slightly more accurate results, however, our much simpler and faster predictor is already sufficient for our method as initialization. The palm and fingers are indexed as C: palm, T: thumb, I: index, M: middle, R: ring, P: pinky, W: wrist.

5.5 Discussion

While our approach is not really biologically-inspired, it should be noted that similar feedback mechanisms also have the support of strong biological evidence. It has been shown that feedback paths in the brain and especially in the visual cortex actually consist of *more* neurons than the forward path [49]. Their functional role remains mostly unexplained [27]; our approach could be a possible explanation in the case of feedback in the visual cortex, but of course, this would need to be proved.

It should also be noted that our predictor and our synthesizer are trained with exactly the same data. One may then ask how our approach can improve the first estimate made by the predictor. The combination of the synthesizer and the updater network provides us with the possibility for simply yet considerably augmenting the training data to learn the update of the pose: For a given input image, we can draw arbitrary numbers of samples of poses through which the updater is then trained to move closer to the ground truth. In this way, we can explore regions of the pose space which are not present in the training data, but might be returned by the predictor when applied to unseen images. Conceptually, it

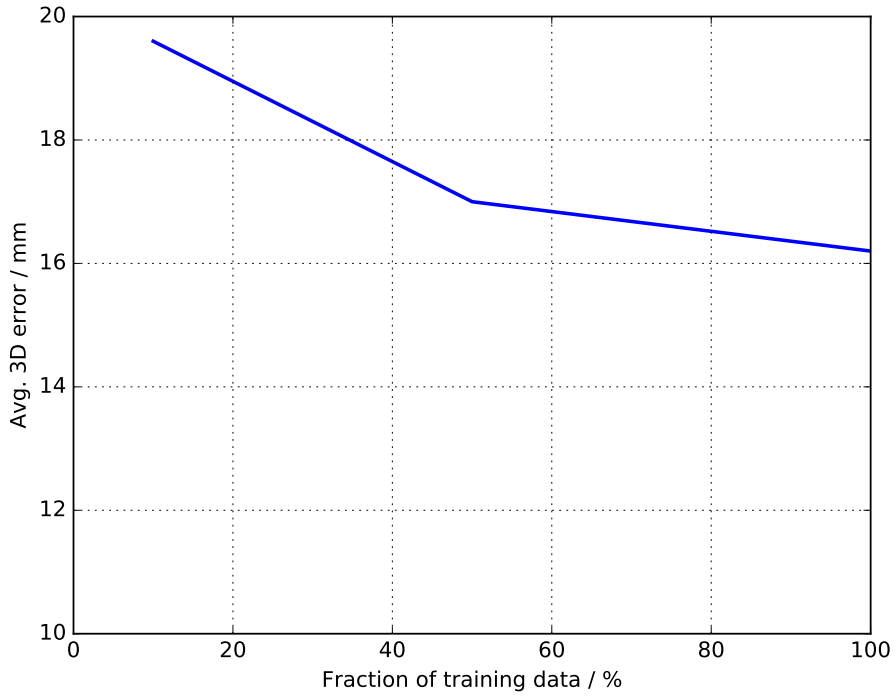


Figure 5.19: Evaluation of number of training samples. We plot the average 3D error over the fraction of training data used. We randomly subsample the training data. The accuracy is only slightly worse when using only half of the training data, but then significantly gets worse.

is easier to add realistic noise to the poses than to the input images.

Finally, our approach has nothing really specific to the hand pose detection or the use of a depth camera, since all its components are learned, from the prediction of the initialization to the generation of images and the update computation. The representation of the pose itself is also very simple and does not have to take into account the specifics of the structure of the hand. We therefore believe that, given proper training data, our approach can be applied to many detection and tracking problems and also with different sensors.

Since the original publication of our feedback method [184] in 2015, our general approach was successfully applied to other fields, such as 3D object pose estimation. [203] predicts updates on 2D object locations. Similarly, [144, 158] use a CAD rendering of an object together with the input image to predict an update for the rendered CAD pose to better resemble the object pose in the input image.

Figure 5.20: Qualitative results on NYU dataset [271]. We show the inferred joint locations on the depth images where the depth is color coded in gray-scale. The individual fingers are color coded, where the bones of each finger share the same color, but with a different hue. The left image of each pair shows the initialization and the right image shows the pose after applying our feedback method. Our method applies to a wide variety of poses and is tolerant to noise, self-occlusions and missing depth values as shown in several images.

Part II

Training Data for 3D Hand Pose Estimation

6 Semi-Automatic Method for Creating Training Data

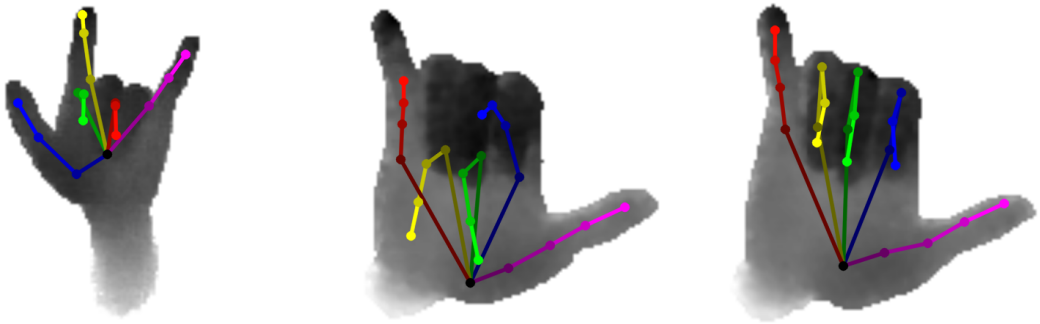
Contents

In the previous part, we introduced methods for accurate and fast 3D hand pose estimation from images. While these methods and many other recent hand pose estimation methods critically rely on a training set of labeled frames, the creation of such a dataset is a challenging task that has been overlooked so far. As a result, existing datasets are limited to a few sequences and individuals, with limited accuracy, and this prevents these methods from delivering their full potential. We propose a semi-automated method for efficiently and accurately labeling each frame of a hand depth sequence with the corresponding 3D locations of the joints: The user is asked to provide only an estimate of the *2D reprojections* of the visible joints in some reference frames, which are automatically selected to minimize the labeling work by efficiently optimizing a sub-modular loss function. We then exploit spatial, temporal, and appearance constraints to retrieve the full 3D poses of the hand over the complete sequence. We show that this data can be used to train a recent state-of-the-art hand pose estimation method, leading to increased accuracy.

6.1 Introduction

Recent work on articulated pose estimation [107, 184, 247, 254, 271] has shown that a large amount of accurate training data makes reliable and precise estimation possible. This observation is also valid for the methods introduced in the previous chapters.

For human bodies, motion capture [107] can be used to generate large datasets with sufficient accuracy. However, creating accurate annotations for hand pose estimation is far more difficult, and still an unsolved problem. Motion capture is not an option anymore, since the fiducials required to track the joints of a hand would corrupt the images. Moreover, the human hand has more Degrees of Freedom (DoF) than are generally considered for 3D body tracking, and an even larger amount of training data is probably required.



(a) **(b)** **(c)**

Figure 6.1: Recent hand pose datasets exhibit significant errors in the 3D locations of the joints. (a) is from the ICVL dataset [254], and (b) from the MSRA dataset [247]. Both datasets were annotated by fitting a 3D hand model, which is prone to converge to a local minimum. In contrast, (c) shows the annotations acquired with our proposed method for the same frame as in (b).

The appearance of depth sensors has made 3D hand pose estimation easier, but has not solved the problem of the creation of training data entirely. Despite its importance, the creation of a training set has been overlooked so far, and authors have had to rely on *ad hoc* ways that are prone to errors, as shown in Figure 6.1. Complex multi-camera setups [12, 241, 271, 277] together with tracking algorithms have typically been used to create annotations. For example, Tompson *et al.* [271] used a complex camera setup with three RGB-D cameras to fit a predefined 3D hand model. Looking closely at the resulting data, it seems that the 3D model was often manually adjusted to fit the sequences better and in between these manually adjusted frames the fit can be poor. Further, the dataset of [254] contains many misplaced annotations, as discussed by [183, 249]. Although recent datasets [247] have paid more attention to high quality annotations, they still contain annotation errors, such as multiple annotations on a single finger, or mixing fingers. These errors result in noisy training and test data, and make training and evaluating uncertain. This issue was addressed recently by [14], which shows that using a robust loss function for training rather than a least-squares one results in better performance.

These problems can be circumvented via using synthetic training data [212, 215, 299]. Unfortunately, this does not capture the sensor characteristics, such as noise and missing

data typical of depth sensors, nor the physical constraints that limit the range of possible hand poses [296]. Another common approach to creating training data is using crowd source platforms, such as Amazon Mechanical Turk. In our case, however, the annotations should be in 3D, which makes the task very challenging if done manually, even with a depth sensor: The sensor can only provide the depth of the skin, not the joints themselves, and even this information is not always available in the case of self-occlusion or missing data. Thus, this task does not lend itself to this kind of crowd sourcing with untrained workers. Moreover, whatever the method, one has to recreate new data for each new sensor.

For all of these reasons, we developed a semi-automated approach that makes it easy to annotate sequences of articulated poses in 3D. The general pipeline of our approach is shown in Figure 6.2. We ask a human annotator to provide an estimate of the 2D reprojections of the visible joints in frames we refer to as *reference frames*. We propose a method to automatically select these reference frames to minimize the annotation effort, based on the appearances of the frames over the whole sequence. We then use this information to automatically infer the 3D locations of the joints for all the frames, by exploiting appearance, temporal, and distances constraints. If this inference fails for some frames, the annotator can still provide additional 2D reprojections; by running the global inference again, a single additional annotation typically fixes many frames.

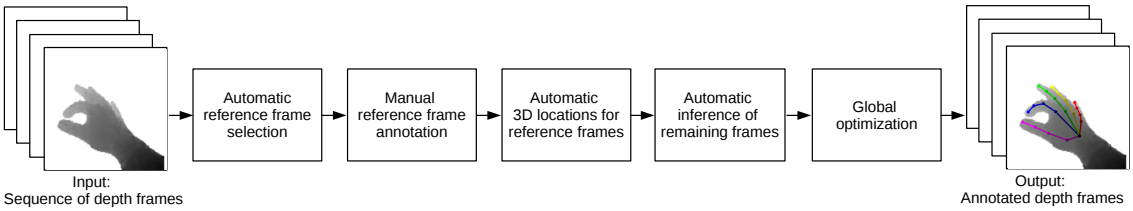


Figure 6.2: General pipeline of the proposed training data generation. See text for details.

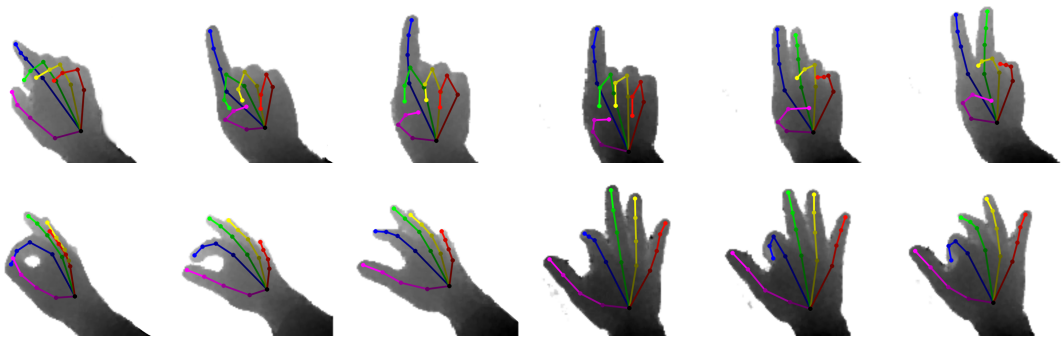


Figure 6.3: Our method made it possible to create a fully annotated dataset of more than 2000 frames from an egocentric viewpoint, which is considered to be very challenging [214].

We evaluate our approach using both synthetic data and real images. We also show that we can improve the annotations of existing datasets, which yield more accurate predicted

poses. As Figure 6.3 shows, our approach also allows us to provide the first fully annotated egocentric sequences, with more than 2000 frames in total.

In the remainder of this chapter, we first give a short review of related work on creating training data in Section 6.2. Then, we introduce our contributions in Section 6.3 and evaluate them in Section 6.4.

6.2 Related Work on Creating Training Data

Various approaches to create training and test data with ground truth for evaluation have been proposed in the literature. Complex camera setups can be used to mitigate problems with self-occlusions. Tompson *et al.* [271] relied on three RGB-D cameras. They used a predefined 3D hand model that had to be manually readjusted for each person. When looking closely at the data, it appears that the dimensions of the model were modified over the sequences, probably to fit the incoming images better. This dataset was taken from a frontal view of the user, which limits the range of the poses. Sridhar *et al.* [241] used five RGB and two RGB-D cameras, and annotated only the finger tips, which is not enough for full 3D pose estimation. [12, 277] required eight RGB cameras to capture hand interactions, however, causing significant restrictions on hand movement within this setup.

An alternative to these complex setups with restricted ranges are single camera approaches. For example, Tang *et al.* [254] used the method from [165] to fit a hand model to a single depth image. Similarly, [201, 247] used a single depth camera to fit a predefined 3D hand model. These methods are based on frame-to-frame tracking. This requires manual supervision and leads to many errors if the optimization does not converge correctly.

Very accurate training data can be generated using synthetic models, as was done in [212, 215, 299] for example. However, synthetic data does not capture the full characteristics of the human hand and sensor characteristics are not considered. [299] added synthetic sensor noise, however, it is difficult to model this in a general way.

There are also invasive methods for acquiring accurate 3D locations. For example, [292] used a sophisticated magnetic tracker but only for finger tips. [299] used a data glove, but unfortunately data gloves are not very accurate and would be visible in the training images, thus biasing learning algorithms.

A different approach was proposed by Yasin *et al.* [306], who matched 2D poses against a set of 3D poses obtained from motion capture sequences, by comparing the 2D poses with the reprojections of the 3D poses in virtual cameras. This is an interesting approach, however, 2D pose estimation is also an open research topic and still prone to errors.

For egocentric 3D hand pose annotation, Rogez *et al.* [214] proposed a semi-automatic labeling method, where a user labels the 2D locations of a few joints, and chooses the closest 3D pose among a set of synthetic training samples. The 3D pose is then estimated from the 2D annotations and the selected 3D training pose. The user then has to manually refine the pose in 3D. This process is iterated until an appealing result is achieved. This is

a time consuming task and thus, they only created a temporally sparse set, which is only sufficient for testing and additional data is required for training.

Semi-automated methods for annotating video sequences like ours are not new to Computer Vision. [139] exploited object silhouettes in reference frames to predict the object silhouettes in the remaining frames. [4] also used manual annotations of some frames to iteratively train a 2D object detector. [291] used annotations in manually selected frames, to predict the annotations of the remaining ones. Compared to these works, we propose a method for selecting the frames to be annotated, minimize manual work, but more importantly, our approach provides a complex articulated 3D structure from 2D annotations.

6.3 Creating Training Data Efficiently

Given a sequence of N depth images $\{\mathcal{D}_i\}_{i=1}^N$ capturing a hand in motion, we want to estimate the 3D joint locations for each $\mathcal{D}_i$ with minimal effort. Our approach starts by automatically selecting some of the depth images we will refer to as *reference frames* (Section 6.3.1). A user is then asked to provide the 2D reprojections of the joints in these reference frames, from which we infer their 3D locations in these frames (Section 6.3.2). We propagate these 3D locations to the other frames (Section 6.3.3), and we perform a global optimization, enforcing appearance, temporal, and spatial constraints (Section 6.3.4).

6.3.1 Selecting the Reference Frames

A simple way to select the reference frames would be to regularly sample the image sequence in time, and select, for example, every tenth frame as reference frame. However, this solution would be sub-optimal: Sometimes the fingers move fast, and a higher sampling rate would be required, while they can also move more slowly, requiring less manual annotation. Moreover, hand motion performers tend to move back to similar poses at wide intervals, and annotating the same poses several times should be avoided.

Simple temporal sampling therefore does not seem to be a good approach. Ideally, we would like to select as few reference frames as possible, while making sure that for each frame, there is at least one reference frame that is similar enough. This will ensure that we can match them together and estimate the joint reprojections in the frame. Let us assume that we know a distance function $d(\mathcal{D}_i, \mathcal{D}_j)$ that can be used to evaluate the similarity between two depth images $\mathcal{D}_i$ and $\mathcal{D}_j$. Then, the reference frame selection can be formulated as the following Integer Linear Program (ILP):

$$\underset{\{x_i\}_{i=1}^N}{\arg\min} \sum_{i}^{N} x_i \quad \text{s.t.} \quad \forall i \sum_{j \in E_i} x_j \geq 1 \,, \tag{6.1}$$

$$\text{with } E_i = \{j \mid d(\mathcal{D}_i, \mathcal{D}_j) \leq \rho\} \,, \tag{6.2}$$

where the x_i indicate which frames are selected as reference frames ($x_i = 1$ if $\mathcal{D}_i$ is selected, and 0 otherwise). E_i is the set of indices of the frames that are similar enough to frame i for matching, according to a distance function $d(\cdot, \cdot)$, and ρ is a threshold.

This formulation guarantees that we find the global optimum. We implemented it using [16] but unfortunately, optimization turned out to be intractable for real problems with the number of frames N larger than about 10^3. Thus, we turned to the suboptimal but tractable approach by optimizing:

$$\max_{\mathcal{R}} \; f(\mathcal{R}) \quad \text{s.t.} \quad |\mathcal{R}| < M \,, \tag{6.3}$$

where $\mathcal{R}$ is the set of selected reference frames, M the maximum number of reference frames, and $f(\mathcal{R})$ is the number of frames within the chosen distance ρ to at least one of the frames in $\mathcal{R}$. In this approach, if M is set too small, some frames may not have a reference frame near them, but we can trade off the coverage by reference frames with the amount of annotation work. This optimization problem is a submodular problem and it is NP-complete, but what makes it attractive is that a simple greedy optimization was shown to deliver a solution that is close to the globally optimal one [177]. This greedy optimization procedure simply proceeds by adding the element e to the set $\mathcal{R}$ that maximizes the difference $f(e \cup \mathcal{R}) - f(\mathcal{R})$ as long as the number of reference frames is smaller than M.

We define the distance function $d(\cdot, \cdot)$ on descriptors computed for depth images. We tried LineMOD [93] and HOG [47]. However, the best results were achieved by cosine distance between low-dimensional embeddings computed by a Convolutional Autoencoder [160]. Figure 6.4 shows several examples of reference frames selected with this method, visualized along with the depth image embedding.

The autoencoder is trained by minimizing the ℓ_2-norm of the reconstruction of the input image. We train the autoencoder in a greedy, layer-wise fashion [96]. The architecture is shown in Figure 6.5. The encoder part of the autoencoder consists of four convolutional layers, each with 32 kernels of size 5×5. Each convolutional layer is followed by a max-pooling layer with a window size of 2×2. The convolutional layers are followed by four fully connected layers. The decoder part is a mirrored version of the encoder, but instead of max-pooling layers, we use unpooling layers [314], which upsample the feature maps by a factor of 2.

6.3.2 Initializing the 3D Joint Locations in the Reference Frames

Once the procedure described in the previous section has selected the reference frames, a human annotator has to label them. The annotator is only required to provide the 2D reprojections of the joints with visibility information in each reference frame, and whether these joints are closer or farther from the camera than the parent joint in the hand skeleton tree. This can be done easily and quickly, and we use this information to

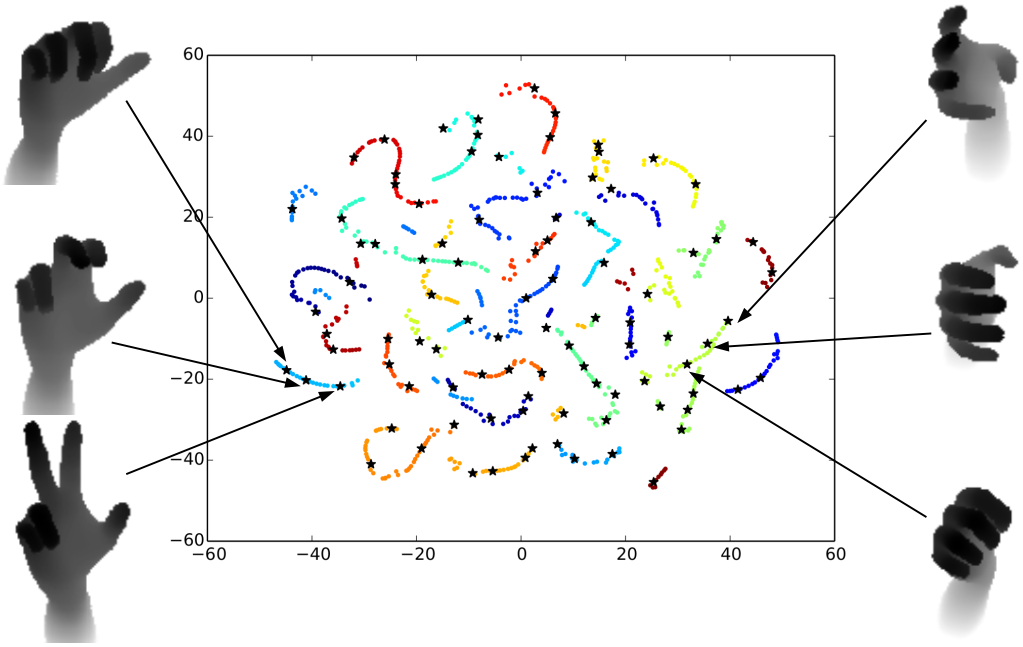


Figure 6.4: t-SNE [280] visualization of the depth image embedding over a sequence. Each colored dot • represents a frame, the color encodes the temporal order. Temporal changes of the hand articulations can be clearly observed from the different trajectories. The reference frames are shown as black stars ★. We automatically select them so that their annotations can be propagated to the other frames while minimizing the manual annotation effort. The selected reference frames cover a maximum of other frames within a distance ρ, based on their appearance. Note that t-SNE sometimes moves points far apart that are close to each other in the original space. This is why the points do not form a continuous curve even if they correspond to consecutive frames [280].

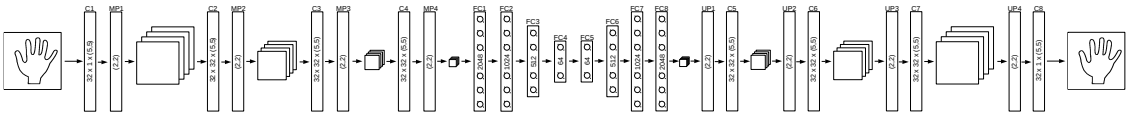


Figure 6.5: Architecture of the Convolutional Autoencoder trained to reconstruct the input image. We use the activations computed after FC4 as embedding to select the reference frames. C denotes a convolutional layer, FC a fully connected layer, UP an unpooling layer, and MP a max-pooling layer. All layers have ReLU activation functions [129].

automatically recover the 3D locations of the joints. It is useful to know the positions of consecutive joints in relation to the camera in order to avoid possible mirroring ambiguities typical of articulated structures [198, 234, 258]. We refer to this information as the *z-order* constraint.

To automatically recover the 3D locations of the joints, we optimize the following

constrained non-linear least squares problem for each reference frame:

$$\begin{aligned} &\arg\min_{\{\mathbf{P}_{r,k}\}_{k=1}^{K}} \sum_{k=1}^{K} v_{r,k} \|\text{proj}(\mathbf{P}_{r,k}) - l_{r,k}\|_2^2 \\ &\text{s.t.} \quad \forall k \;\; \|\mathbf{P}_{r,k} - \mathbf{P}_{r,p(k)}\|_2^2 = d_{k,p(k)}^2 \\ &\qquad \forall k \;\; v_{r,k} = 1 \Rightarrow \mathcal{D}_r[l_{r,k}] < z(\mathbf{P}_{r,k}) < \mathcal{D}_r[l_{r,k}] + \epsilon \\ &\qquad \forall k \;\; v_{r,k} = 0 \Rightarrow z(\mathbf{P}_{r,k}) > \mathcal{D}_r[l_{r,k}] \\ &\qquad \forall k \;\; (\mathbf{P}_{r,k} - \mathbf{P}_{r,p(k)})^\top \cdot c_{r,k} > 0 \end{aligned} \tag{6.4}$$

where r is the index of the reference frame. $v_{r,k} = 1$ if the k-th joint is visible in the r-th frame, and 0 otherwise. $\mathbf{P}_{r,k}$ is the 3D location of the k-th joint for the r-th frame. $l_{r,k}$ is its 2D reprojection as provided by the human annotator. proj($\mathbf{P}$) returns the 2D reprojection of a 3D location. $p(r)$ returns the index of the parent joint of the k-th joint in the hand skeleton. $d_{k,p(k)}$ is the known distance between the k-th joint and its parent $p(k)$. $\mathcal{D}_r[l_{r,k}]$ is the depth value in $\mathcal{D}_r$ at location $l_{r,k}$. $z(\mathbf{P})$ is the depth of 3D location $\mathbf{P}$. ϵ is a threshold used to define the depth interval of the visible joints. In practice, we use $\epsilon = 15\,\text{mm}$ given the physical properties of the hand. $c_{r,k}$ is equal to the vector $[0, 0, -1]^\top$ if the k-th joint is closer to the camera than its parent in frame r, and $[0, 0, 1]^\top$ otherwise. $(\mathbf{P}_{r,k} - \mathbf{P}_{r,p(k)})$ is the vector between joint k and its parent in this frame.

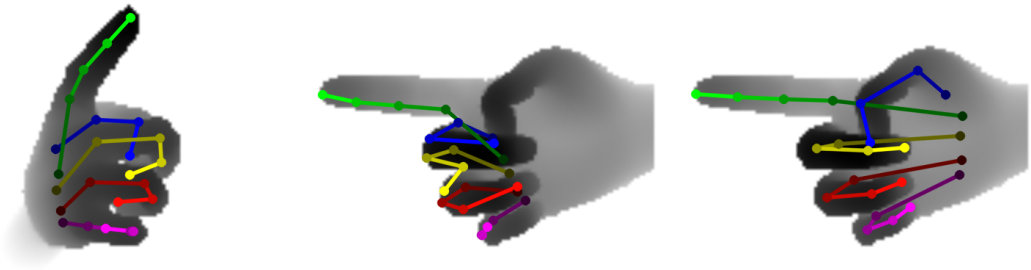
Together, the terms of Eqn. (6.4) assure that: (1) the bone lengths of the skeleton are respected; (2) visible joints are in range of observed depth values; (3) hidden joints are not in front of observed depth values; and (4) depth order constraints of parent joints are fulfilled. We currently assume that the lengths $d_{k,p(k)}$ are known. In practice, we measure them from a depth image of the hand with open fingers and parallel to the image plane. It may also be possible to optimize these distances as they are constant over the sequences from the same person.

We optimize this problem with SLSQP [127]. Equality constraints are hard to optimize, so we relax them and replace the constraints by a term in the loss function that penalizes constraint violations. We use a simple scheduling procedure to progressively increase the weight of this term. This gives us a reasonable initial estimate of the 3D pose of the hand for each reference frame.

We initialize the joint depth with the measurement from the depth sensor at the annotated 2D location. This is shown in Figure 6.6, which depicts the initialized 3D locations and the result after optimizing the relaxation of Eqn. (6.4).

6.3.3 Initializing the 3D Joint Locations in the Remaining Frames

The previous section described how to compute a first estimate for the 3D locations of the joints in the reference frames. Next, we iteratively propagate these 3D locations from the reference frames to the remaining frames, in a way similar to [131], as explained in this



(a) 2D annotation. **(b)** 3D initialization. **(c)** 3D result.

Figure 6.6: Optimization steps for reference frames. We start with the 2D annotations on a depth image provided by a user (a), and backproject them to initialize their 3D location estimates. (b) shows the same pose rendered from a different viewpoint, depicting the depth initialization of the joints. We then optimize the constrained non-linear least squares loss of Eqn. (6.4) on these initial 3D locations. The result is shown in (c), again rendered from a different viewpoint, but now with better aligned 3D locations.

section. This gives us an initialization for the joint locations in all the frames. The next subsection will explain how we refine them in a global optimization procedure.

$\mathcal{I}$ is used to denote the set of frames for which the 3D locations of the joints have already been initialized. At the beginning, $\mathcal{I}$ is initialized to the set of reference frames, but each time we estimate the joints for a frame, this frame is added to $\mathcal{I}$. At each iteration, a frame $\hat{c}$ not yet initialized and its closest frame $\hat{a} \in \mathcal{I}$ are selected:

$$\begin{bmatrix} \hat{c} \\ \hat{a} \end{bmatrix} = \underset{\substack{c \in [1;N] \setminus \mathcal{I} \\ a \in \mathcal{I}}}{\arg\min}\, d(\mathcal{D}_c, \mathcal{D}_a)\ . \tag{6.5}$$

We use the appearance of the joints in $\hat{a}$ to predict their 3D locations $\{\mathbf{P}_{\hat{c},k}\}_k$ in $\hat{c}$ by minimizing:

$$\begin{aligned} &\underset{\{\mathbf{P}_{\hat{c},k}\}_k}{\arg\min} \sum_k \mathrm{ds}(\mathcal{D}_{\hat{c}}, \mathrm{proj}(\mathbf{P}_{\hat{c},k}); \mathcal{D}_{\hat{a}}, l_{\hat{a},k})^2 \\ &\text{s.t.} \quad \forall k \quad \|\mathbf{P}_{\hat{c},k} - \mathbf{P}_{\hat{c},p(k)}\|_2^2 = d_{k,p(k)}^2\ , \end{aligned} \tag{6.6}$$

where $\mathrm{ds}(\mathcal{D}_1, \mathrm{proj}(\mathbf{P}_1); \mathcal{D}_2, l_2)$ denotes the dissimilarity between the patch in $\mathcal{D}_1$ centered on the projection $\mathrm{proj}(\mathbf{P}_1)$ and the patch in $\mathcal{D}_2$ centered on l_2. This optimization looks for joints based on their appearances in frame $\hat{a}$ while enforcing the 3D distances between the joints. We use the Levenberg-Marquardt algorithm to solve Eqn. (6.6), by relaxing the hard constraint with a weighted additional term in the loss function.

As illustrated in Figure 6.7, we initialize the optimization problem of Eqn. (6.6) by

aligning frames $\hat{c}$ and $\hat{a}$ using SIFTFlow [150]. This maps the 2D reprojections of the joints in frame $\hat{a}$ to 2D locations $\{\tilde{l}_k\}_k$ in frame $\hat{c}$. We backproject each $\tilde{l}_k$ on the depth image $\mathcal{D}_{\hat{c}}$ to initialize $\mathbf{P}_{\hat{c},k}$. If the depth information is missing at $\tilde{l}_k$, we use the 3D point that reprojects on $\tilde{l}_k$ and with the same depth as $\mathbf{P}_{\hat{a},k}$.

(a) Closest reference frame. **(b)** Initialization with SIFTFlow. **(c)** After optimization.

Figure 6.7: Initialization of locations on a non-reference frame. (a) shows the 3D locations for the closest reference frame. (b) We propagate the 3D locations using SIFTFlow [150]. (c) 3D locations after optimization of Eqn. (6.6).

6.3.4 Global Optimization

The previous optimization already gives already a good estimate for the 3D locations of the joints in all frames. However, each frame is processed independently. We can improve the estimates further by introducing temporal constraints on the 3D locations. We therefore perform a global optimization over all the 3D locations $\mathbf{P}_{i,k}$ for all the frames by minimizing:

$$\sum_{i\in[1;N]\setminus\mathcal{R}} \sum_{k} \mathrm{ds}(\mathcal{D}_i, \mathrm{proj}(\mathbf{P}_{i,k}); \mathcal{D}_{\hat{i}}, l_{\hat{i},k})^2 + \tag{C}$$

$$\lambda_M \sum_{i} \sum_{k} \|\mathbf{P}_{i,k} - \mathbf{P}_{i+1,k}\|_2^2 + \tag{TC}$$

$$\lambda_P \sum_{r\in\mathcal{R}} \sum_{k} v_{r,k} \|\mathrm{proj}(\mathbf{P}_{r,k}) - l_{r,k}\|_2^2 \tag{P}$$

$$\text{s.t.} \quad \forall i,k \quad \|\mathbf{P}_{i,k} - \mathbf{P}_{i,p(k)}\|_2^2 = d_{k,p(k)}^2 \,.$$

The first term (C) sums the dissimilarities of the joint appearances with those in the closest reference frame $\hat{i} = \arg\min_{a\in\mathcal{R}} d(\mathcal{D}_i, \mathcal{D}_a)$ over the non-reference frames i. The second term (TC) is a simple 0-th order motion model that enforces temporal smoothness of the 3D locations. The last term (P) of the sum ensures consistency with the manual

2D annotations for the reference frames. λ_M and λ_P are weights to trade off the different terms.

This is a non-convex problem, and we use the estimates from the previous subsection to initialize it. This prevents the optimization from falling into bad local minimums.

This problem has $3KN$ unknowns for K joints and N frames. In practice, the number of unknowns varies from 10^5 to 10^7 for the datasets we consider in the evaluation. Fortunately, this is a sparse problem, which can be efficiently optimized with the Levenberg-Marquardt algorithm.

6.3.5 Relaxations of Optimization Problems

We show the relaxations of the optimization problems presented earlier. For all problems, we use the penalty method [180] for equality constraints. For this, we introduce a multiplier that penalizes constraint violations.

The given optimization problem in Eqn. (6.4) includes an equality and inequality constraints. By introducing multipliers μ and β, the relaxation is:

$$\begin{aligned}
\underset{\{\mathbf{P}_{r,k}\}_{k=1}^{K}}{\arg\min} & \sum_{k=1}^{K} v_{r,k} \|\text{proj}(\mathbf{P}_{r,k}) - l_{r,k}\|_2^2 + \mu^b \sum_k \left(\|\mathbf{P}_{r,k} - \mathbf{P}_{r,p(k)}\|_2^2 - d_{k,p(k)}^2 \right)^2 + \\
& \beta \sum_{\{k|v_{r,k}=1\}} \min\left((\mathbf{P}_{r,k} - \mathbf{P}_{r,p(k)})^\top \cdot c_{r,k}, 0 \right)^2 \qquad (6.7) \\
\text{s.t.} \quad & \forall k \quad v_{r,k} = 1 \Rightarrow \mathcal{D}_r[l_{r,k}] < z(\mathbf{P}_{r,k}) < \mathcal{D}_r[l_{r,k}] + \epsilon \\
& \forall k \quad v_{r,k} = 0 \Rightarrow z(\mathbf{P}_{r,k}) > \mathcal{D}_r[l_{r,k}]
\end{aligned}$$

We start with $\mu^0 = 10^0$ and increase $\mu^{b+1} = 10\mu^b$ for 5 times. For each μ we solve the problem until convergence, *i.e.*, the decrease of the objective function is smaller than 10^{-5}. Further, we keep $\beta = 1$ fixed. We use the SLSQP algorithm [127] to solve the relaxed problem. The remaining two constraints can be integrated as box constraints in SLSQP.

We relax the equality constraint of the optimization problem in Eqn. (6.6) by introducing a multiplier γ:

$$\underset{\{\mathbf{P}_{\hat{c},k}\}_k}{\arg\min} \sum_k \text{ds}(\mathcal{D}_{\hat{c}}, \text{proj}(\mathbf{P}_{\hat{c},k}); \mathcal{D}_{\hat{a}}, l_{\hat{a},k})^2 + \gamma^b \sum_k \left(\|\mathbf{P}_{\hat{c},k} - \mathbf{P}_{\hat{c},p(k)}\|_2^2 - d_{k,p(k)}^2 \right)^2 . \qquad (6.8)$$

We start with $\gamma^0 = 10^1$ and increase $\gamma^{b+1} = 10\gamma^b$ for 5 times. For each γ we solve the problem until convergence using the Levenberg-Marquardt [140] algorithm.

The equality constraint of the optimization problem in Eqn. (6.7) is relaxed by intro-

ducing a multiplier η:

$$\begin{aligned}
&\sum_{i\in[1;N]\setminus\mathcal{R}}\sum_{k}\mathrm{ds}(\mathcal{D}_i,\mathrm{proj}(\mathbf{P}_{i,k});\mathcal{D}_{\hat{i}},l_{\hat{i},k})^2 + \\
&\lambda_M\sum_{i}\sum_{k}\|\mathbf{P}_{i,k}-\mathbf{P}_{i+1,k}\|_2^2 + \\
&\lambda_P\sum_{r\in\mathcal{R}}\sum_{k}v_{r,k}\|\mathrm{proj}(\mathbf{P}_{r,k})-l_{r,k}\|_2^2 + \\
&\eta^b\sum_{i}\sum_{k}\left(\|\mathbf{P}_{i,k}-\mathbf{P}_{i,p(k)}\|_2^2-d_{k,p(k)}^2\right)^2 .
\end{aligned}$$

We start with $\eta^0 = 10^2$ and increase $\eta^{b+1} = 10\eta^b$ for 5 times. The relaxed problem is solved with sparse Levenberg-Marquardt [140] algorithm.

6.4 Evaluation

To validate our method, we first evaluate it on a synthetic dataset, which is the only way to have depth images with ground truth 3D locations of the joints. We then provide a qualitative evaluation on real images and on the recent MSRA dataset [247]. Finally, we show that we can use our method to create a large dataset of egocentric annotated frames.

6.4.1 Evaluation on Synthetic Data

We used the publicly available framework of [212] to generate synthetic depth images along with the corresponding ground truth annotations. The sequence consists of 3040 frames and shows a single hand performing various poses and arm movements.

Reference Frame Selection In Figure 6.8, we plot the fraction of frames for which the maximum 3D distance of their joints to their locations in the assigned reference frame is lower than a threshold. More formally, we plot the function $n(\tau)$ with:

$$n(\tau) := \frac{1}{N}\left|\left\{i\in[1;N]\mid \max_k\|\mathbf{P}_{i,k}^{\mathrm{GT}}-\mathbf{P}_{a(i),k}^{\mathrm{GT}}\|<\tau\right\}\right|, \tag{6.9}$$

where the $\mathbf{P}_{i,k}^{\mathrm{GT}}$ are the ground truth 3D locations for the joints, and $a(i) = \arg\min_{a\in\mathcal{R}} d(\mathcal{D}_i,\mathcal{D}_a)$. This metric allows us to check if the selection based on the visual appearance using distance $d(\cdot,\cdot)$ also retrieves reference frames that are close in 3D. This is an important factor for the rest of the method, as the propagation step will perform better if a frame is not too far away from the closest reference frame. We use $\rho = 0.1$, which we obtained by cross-validation, however, the reference frame selection is not very sensitive to the exact value.

We compare our selection with a straightforward selection based on regular temporal sampling using the same number of reference frames. According to this metric, our method is significantly better, and it actually yields more accurate 3D annotations: Using our proposed selection, the average error is 5.53 mm, compared to 6.45 mm when taking equitemporal samples. Additionally, the required number of manual reannotations is much higher. Our method required 133 additional 2D locations, compared to 276 manual interventions for the equitemporal selection.

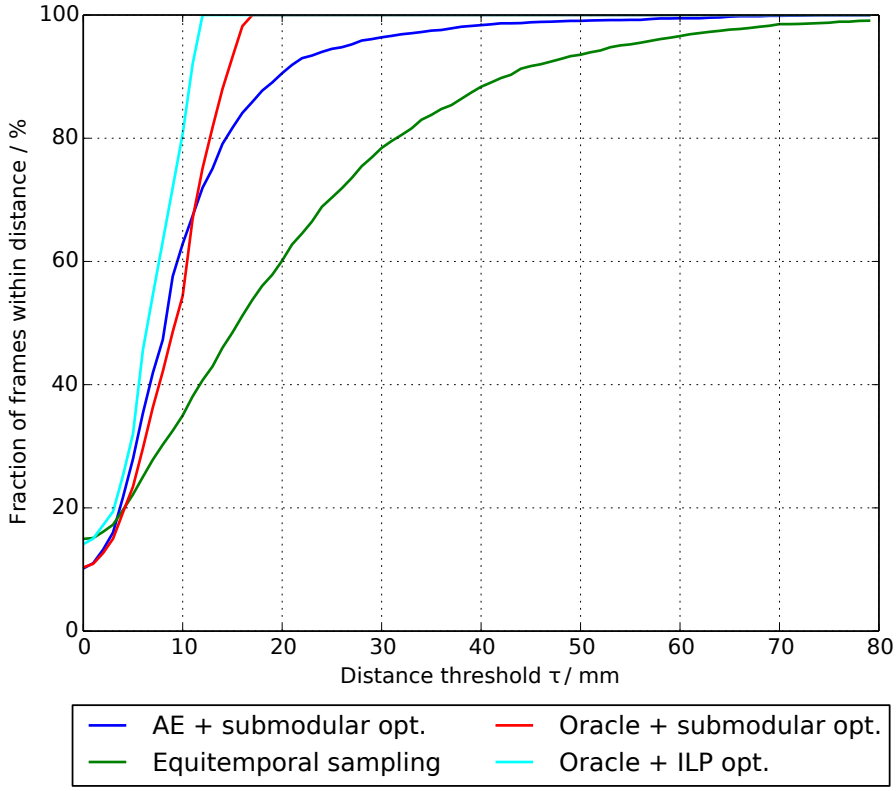


Figure 6.8: Evaluation of the reference frame selection method on the synthetic sequence. We plot the metric $n(\tau)$ of Eqn. (6.9) for our selection of 304 reference frames, and the same number of reference frames sampled regularly in time. With our method, the non-reference frames are closer in 3D to the reference frames. For the oracle, we assume the pose given, and apply the submodular optimization on the Euclidean pose distances. The average pose distance of our proposed selection is only 0.5 mm worse than the oracle. For this sequence, the number of frames is about 10^3, thus solving the *ILP* is still feasible. The approximation using our submodular optimization is close to the optimal *ILP* solution, which on average is only 1 mm better.

Reference Frame Initialization Table 6.1 provides the accuracy of the initialization of the 3D joint locations for the reference frames, and evaluates the influence of the different terms in Eqn. (6.4). We first use the perfect 2D locations of the joints from ground truth.

Method	Visible joints Avg. / median	All joints Avg. / median
2D locations	12.86 / 8.96 mm	19.98 / 13.29 mm
2D & visibility	3.94 / 3.18 mm	6.20 / 3.41 mm
2D & vis & z-order	**2.97 / 2.93 mm**	**3.65 / 2.98 mm**
All + 2D noise	3.70 ± 0.71 mm / 2.59 ± 0.21 mm	4.29 ± 0.63 mm / 2.56 ± 0.23 mm

Table 6.1: Accuracy of reference frame initialization on the synthetic sequence. We provide the average and the median Euclidean joint error over all joints. The highest accuracy can be achieved by combining all of our proposed clues: 2D reprojection errors, visibility, and z-order. The last row shows the robustness of our method to noise, after adding Gaussian noise to the 2D locations.

When only minimizing the 2D reprojection errors with the 3D distance constraints, the error is quite large. Adding the visibility constraint significantly improves the accuracy, but the maximum error is still large. By adding the z-order term, depth ambiguities due to mirroring can be resolved, and the errors get much smaller.

In practice, the 2D locations of the joints provided by a human annotator are noisy. Thus, we evaluated the robustness of our algorithm by adding Gaussian noise with zero mean and a standard deviation of 3 pixels to the 2D locations. For reference, the width of the fingers in the depth image is around 25 pixels. We show the average errors in Table 6.1 after 10 random runs. The errors are only 0.7 mm larger than without noise, which shows the robustness of our method to noisy annotations. Interestingly, the median error is lower with noisy initialization, which can be attributed to the non-convex optimization. Due to noise, the convergence can lead to different local minima, thus improving some joint estimates, but significantly worsening others.

3D Location Propagation and Global Optimization We evaluate the contributions of the different optimization steps in Table 6.2. We implement $\mathrm{ds}(\cdot)$ as normalized cross-correlation with a patch size of 25 pixels. A cross-validation among different correlation methods and different patch sizes has shown that our method is not sensitive to this choice. Further, we use $\lambda_M = 1$ and $\lambda_P = 100$. The choice of these values is not crucial for a good result, as long as $\lambda_P > \lambda_M$ to emphasize the reprojections on the 2D annotations.

If the deviation of the 2D location of a joint gets too large, *i.e.*, the 2D location is more than 5 pixels away from the ground truth location, manual intervention is required. For the synthetic dataset, it was required to readjust the 2D locations of 133 joints in 79 frames, or 0.06% of the total number of joints. Figure 6.9 gives a more exhaustive evaluation of the influence of the chosen number of reference frames. We can obtain a very good accuracy by annotating only a small percentage of the reference frames and correcting an even smaller percentage of joints.

Method	Avg. / median error
Closest reference	11.50 / 5.58 mm
Aligned with SIFTFlow	11.40 / 5.40 mm
Frame optimization	5.76 / 4.34 mm
Global optimization	5.53 / 4.23 mm

Table 6.2: Accuracy of the different stages on the synthetic sequence. We report the average and median Euclidean 3D joint errors. We use the 3D locations of the reference frame to initialize the remaining frames. The first row shows the accuracy if the 3D locations of the closest reference frame are used. The next row shows the contribution of the alignment with SIFTFlow, and the further optimization on the 3D locations. The last row denotes the accuracy after the global optimization. The gain in accuracy with SIFTFlow is small, as it only provides an offset in 2D, but it is useful to make the correlation term contribute properly.

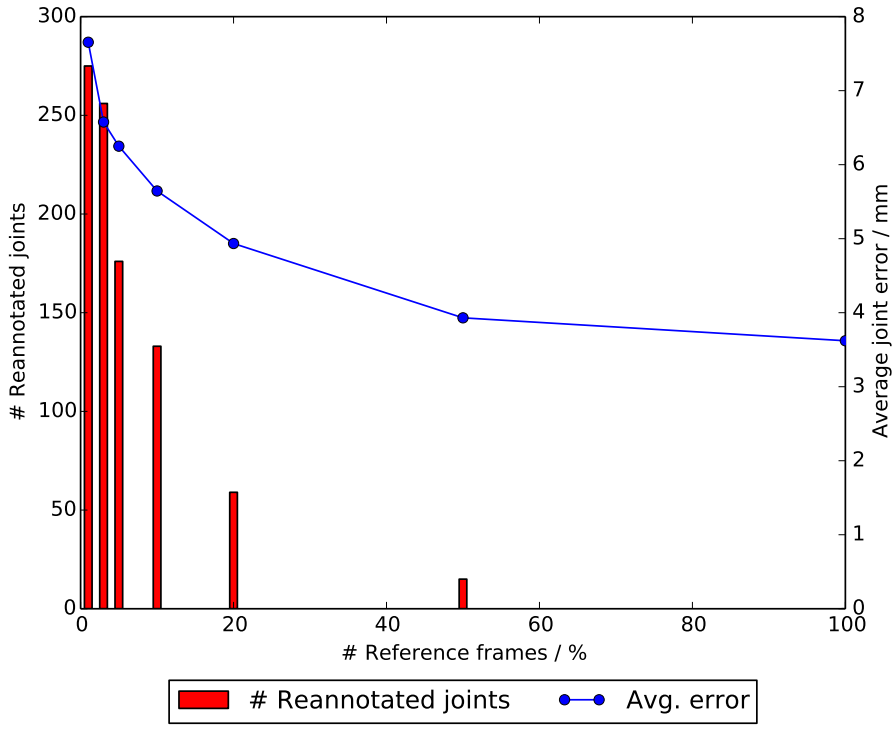
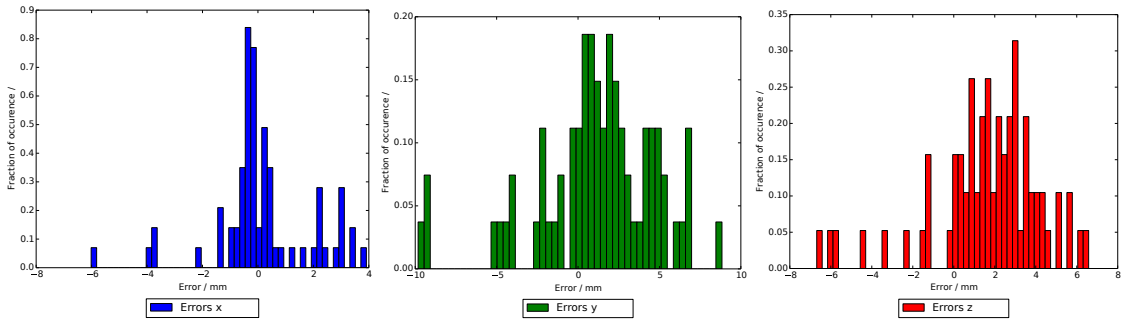


Figure 6.9: Accuracy over the number of reference frames on the synthetic sequence. We plot the average 3D joint error and the number of required additional annotations over the number of initially selected reference frames. The best result can be achieved, when providing manual annotations for all frames, however, more reference frames require more annotation work, and for larger sequences this can be infeasible, *i.e.*, providing manual annotations for about 23k joints for this sequence. When decreasing the number of initial reference frames, the additional annotations of individual joints during the process increases, but only in the hundreds. Using for example 3% of all frames as reference frames requires annotating only 700 joint locations and revising another 250, while still retaining an average 3D annotation error of only 6.5 mm.

Error Analysis In Figure 6.10 we show the distribution of the errors in x-, y-, z-coordinates for the synthetic sequence. The errors in all coordinates are similarly distributed, thus there is no error bias towards a certain coordinate.



(a) **(b)** **(c)**

Figure 6.10: Evaluation of error distributions. We show the distance between our inferred joint locations and the ground truth joint locations for the x-, y-, and z-coordinate.

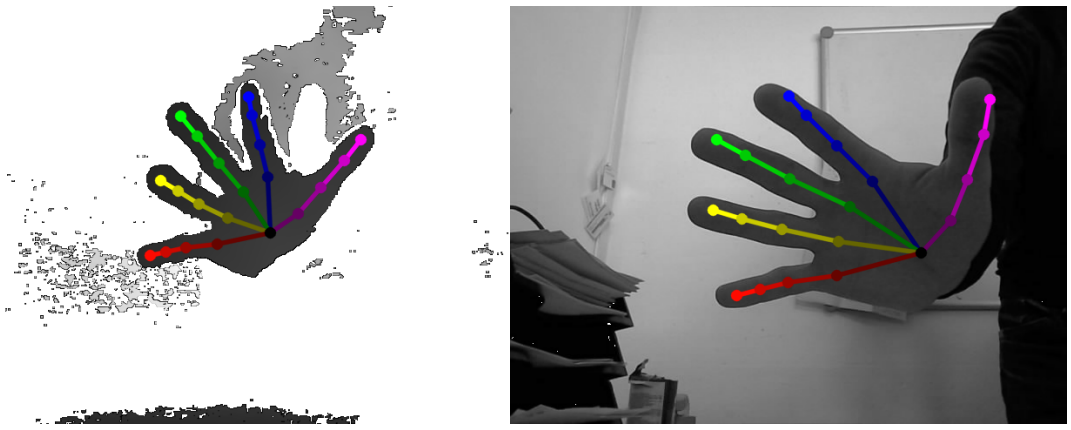
6.4.2 Evaluation on Real Data

To also evaluate real data, we tested our method on a calibrated camera setup consisting of a depth camera and an RGB camera capturing the hand from two different perspectives. We create 3D annotations using the depth camera and project them into the RGB camera. We can then visually check the projections of the annotations. Figure 6.11 shows one example: The joint locations project nearby the real joint locations, which indicates that not only the image coordinates are correct, but also the depth.

6.4.3 Application to the MSRA Dataset

We applied our approach to the MSRA dataset [247], which is currently the largest dataset for hand pose estimation from single depth images. The authors used a state-of-the-art 3D model-based method [201] to obtain the annotations. As discussed earlier, these annotations are not perfect. We used our method to select 10% of the frames (849 out of 8499 for the first subject) as reference frames and manually provided the 2D locations, visibility, and z-order of the joints for these reference frames. We further show a qualitative comparison, and that the higher annotation accuracy leads to better pose estimates, when training the state-of-the-art 3D hand pose estimator DeepPrior that we introduced in Chapter 4.

Qualitative Comparison As "real" ground truth is not available, a direct evaluation is not possible. Figure 6.12 compares different frames for which the distances between



(a) Depth camera. **(b)** Projected into RGB camera.

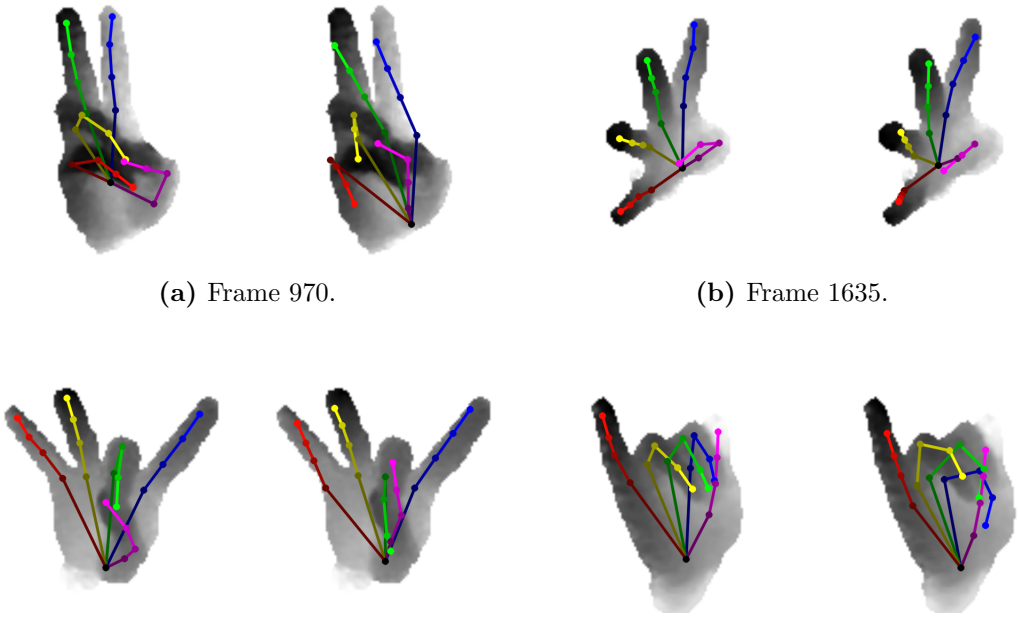
Figure 6.11: Sample frames of our two camera setup. We capture hand articulations from a depth and an RGB camera and apply our method on the depth camera to obtain 3D annotations. These annotations are shown in (a). To evaluate the accuracy, we project the estimated 3D annotations into the RGB camera, which is shown in (b).

the annotations are large. Our annotations appear systematically better. This strongly suggests that our annotations are more accurate over the sequence.

Higher Annotation Accuracy Leads to Better Pose Estimators We further show that better annotations improve the accuracy of state-of-the-art 3D hand pose estimation methods. We train DeepPrior with the original annotations and compare it with the estimator trained using the annotations we obtained with our method. For the evaluation, we perform 10-fold cross-validation, because no explicit test set is specified [247]. The results of this experiment are shown in Figure 6.13. The estimator trained with our annotations converges faster, but to similar average joint errors in the end. This indicates that training is easier when the annotations are better. Otherwise, the estimator may focus on difficult, possibly wrongly annotated samples. The results clearly show that accurate training data is necessary to perform accurate inference. The estimator achieves test set errors of 5.58 ± 0.56 mm using our annotations for training and testing, and 6.41 ± 2.05 mm using the provided annotations. When we train the pose estimator on the provided annotations but evaluate it on our own annotations, the error is 13.23 ± 6.98 mm, which indicates discrepancy among the annotations. However, the visual comparison—which is the best that can be done on real data—shows that our annotations are more accurate.

6.4.4 New Egocentric Dataset

Egocentric 3D hand pose estimation is an appealing feature for different Augmented Reality (AR) or Human-Computer Interaction (HCI) applications. Creating datasets for this



(a) Frame 970. **(b)** Frame 1635.

(c) Frame 3626. **(d)** Frame 8495.

Figure 6.12: Qualitative comparison of the annotations obtained using our method (left image) and the annotations of [247] (right image) on the MSRA dataset [247]. We selected several frames with large differences between the two annotations. Note that the shown sample frames are *not* manually annotated reference frames.

task is very difficult [214]. Egocentric views show severe self-occlusions as fingers are often occluded by the hand and egocentric cameras have a limited field-of-view. Both facts result in a less reliable tracking. Even with manual initialization, fingers are frequently occluded and the hand can move outside the camera view frustum.

We provide a new dataset consisting of more than 2000 frames of several egocentric sequences, each starting and ending with a neutral hand pose and showing a user performing a single or various hand articulations per sequence. We annotated the dataset using our method. In contrast, the 3D model-based implementation of [271] often failed by converging to different local minima, and thus resulted in time consuming fiddling with the model parameters.

We establish a baseline on this dataset, by running two state-of-the-art methods: (1) DeepPrior that we introduced in Chapter 4, which was shown to be among the best methods for third person hand pose estimation [249], and (2) the method of Supancic *et al.* [249], which was initially proposed for hand pose estimation, but especially for hand-object interaction in egocentric views. We perform 5-fold cross-validation and report the average and standard deviation over the different folds. We report the results in Table 6.3. The method of Supancic *et al.* has larger errors, mostly due to flipping ambiguities. For the oracle, we assume the 3D poses known, and return the pose with the smallest Euclidean

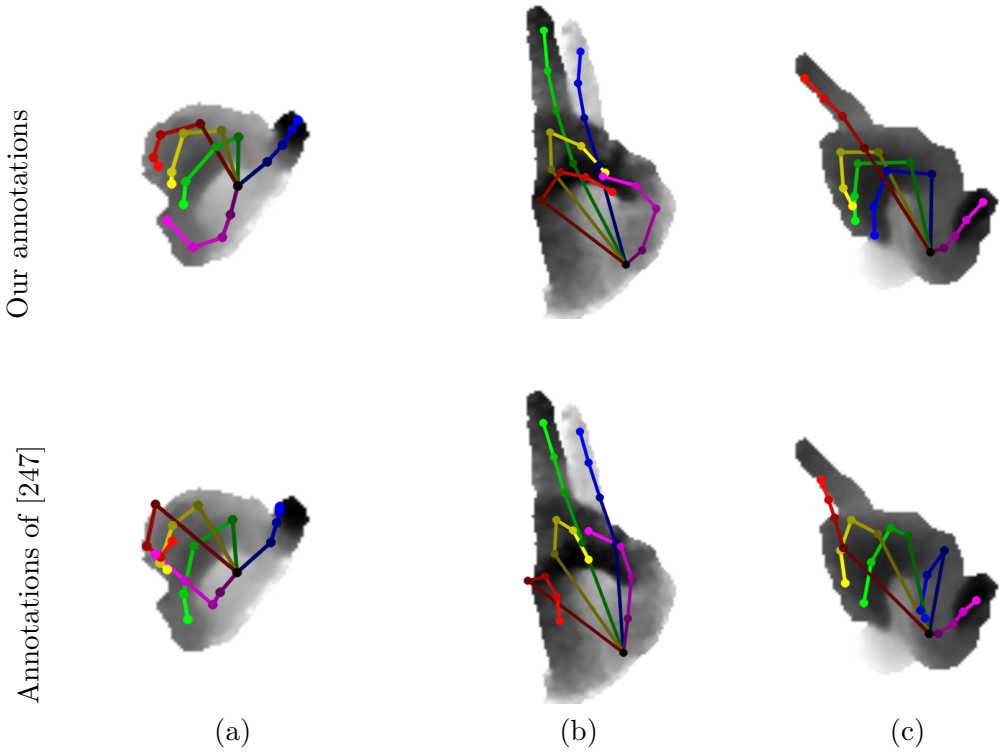


Figure 6.13: Training and testing a 3D hand pose estimator with different annotations for the MSRA dataset [247]. We train our 3D hand pose estimator DeepPrior with the provided annotations and with our revised annotations. We then compare the predicted annotations on a test set. The predictions with our annotations are on the left side. It shows that the estimator learns the incorrect annotations, which leads to inaccurate locations for the test samples. Using our more accurate annotations leads to more accurate results. In (a) note the annotation of the thumb, in (b) the annotations of the pinky and ring fingers, and in (c) the articulation of the index finger.

distance from the training set.

6.4.5 User Evaluation

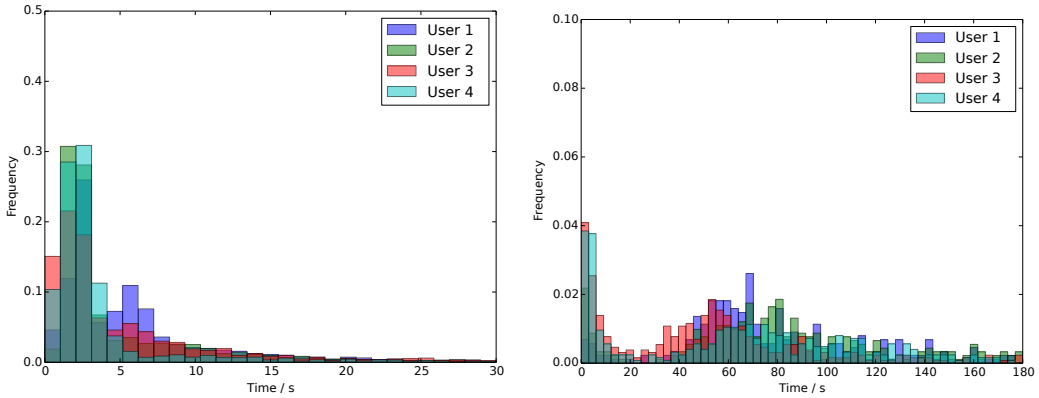
We evaluate the performance of different users using the proposed annotation method to annotate a sequence. We required the users to annotate the synthetic sequence presented in Section 6.4.1 in order to have ground truth labels for a comparison of the accuracy. We evaluate the user performance in terms of 3D pose accuracy and annotation time per frame.

First, the hand needed to be localized in 3D space within the depth image. The time required for this task is shown in Figure 6.14a. The average time is around 2 s per frame and consistent between the users. The long tail of the distribution can be attributed to

Method	Avg. / median error
DeepPrior (Chapter 4)	24.58 ± 16.08 / 19.53 mm
Supancic *et al.* [249]	33.09 ± 21.66 / 26.20 mm
Oracle	20.20 ± 10.92 / 19.47 mm

Table 6.3: Average accuracy on the egocentric hand dataset with 5-fold cross-validation. We apply two state-of-the-art methods to the dataset and report the Euclidean 3D joint errors. For the orcle, we calculate the distance to the nearest sample in the training set.

breaks of the users or harder examples that required more time.[1]
Second, the 3D pose in terms of 21 joint locations needed to be annotated. The time for this task is shown in Figure 6.14b. The average time needed for this task was around 1 min per frame. However, the distribution shows that there are many frames that required less than 20 s. Optimization of Eqn. (6.4) was around 1 s and had to be run after each adjustment of the 2D locations that resulted in additional computation time that is included in this statistics.



(a) Annotation time for hand detection. **(b)** Annotation time for hand pose.

Figure 6.14: Annotation time for hand detection in (a) and for hand pose in (b).

Further, we evaluate the accuracy of the user annotations. First, we evaluate the accuracy of the 3D hand localization. We plot the fraction of frames where the 3D error is below a specific threshold in Figure 6.15. Four users annotated each frame using our proposed method with the 3D hand location. The evaluation shows, that all users marked all the 3D hand locations with less than 20 mm error.

Next, we evaluate the accuracy of the hand pose annotations. Therefore, we evaluate

[1]Due to privacy reasons, we did only track the time per frame, not the actual clicks and actions the users perform.

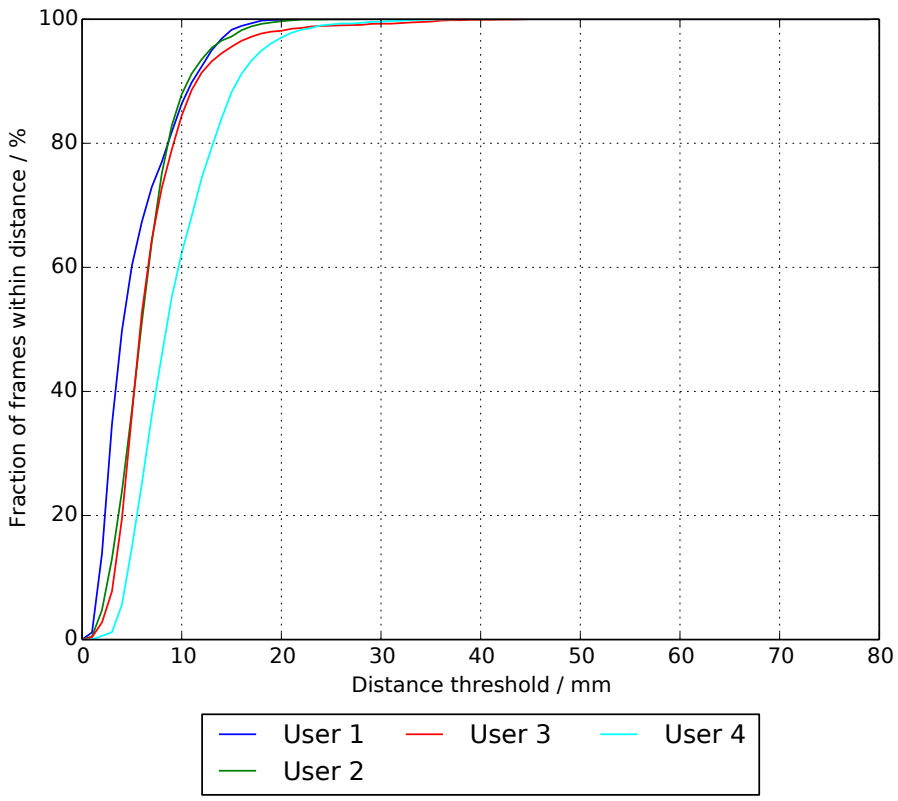
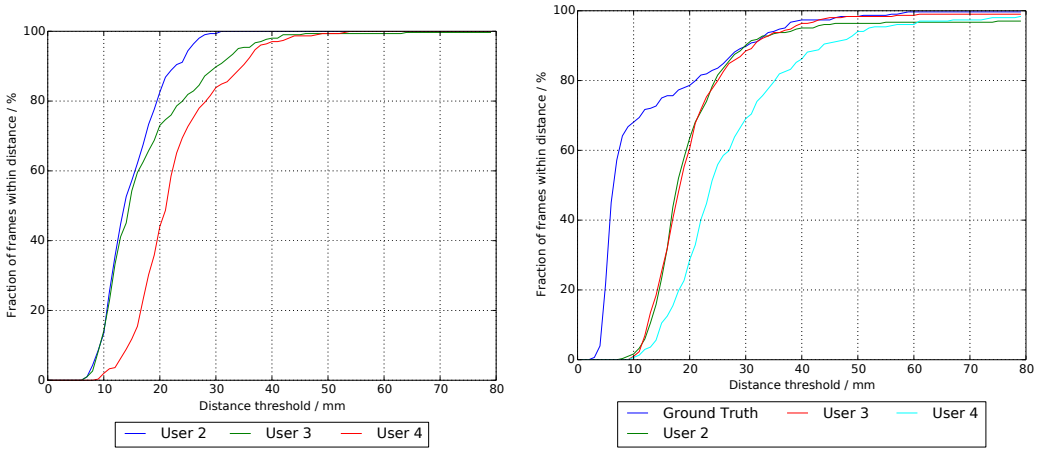


Figure 6.15: Accuracy of 3D hand localization. Four users manually annotated the 3D hand locations of the synthetic sequence. We plot the fraction of frames where the 3D error is below a specific threshold.

the 2D user locations of three users and the inferred 3D hand pose using the method presented in Section 6.3.2. Again, we plot the fraction of frames where the maximum error per frame is below a specific threshold in Figure 6.16. Around 80% of the 2D annotations are below 20 px error, mostly due to inaccurate occluded joints. One can observe that there is a larger discrepancy in terms of accuracy between the different users, which can be attributed to their individual skills.

The errors in the 3D locations are due to the inaccurate 2D user annotations, since there are much more high accuracy frames when using the ground truth 2D locations. A study of Supancic *et al.* [249] comes to similar conclusion in terms of accuracy. However, their users where provided color and depth image together with a 3D hand model for visual feedback, which required to fit inverse kinematics to the annotations. In our case, the users only used depth images without any 3D hand model and inverse kinematics.



(a) Joint annotation in 2D. **(b)** Joint annotation in 3D.

Figure 6.16: Accuracy of joint annotation in 2D and 3D. Three users manually annotated the 2D joint locations of the synthetic sequence. We plot the fraction of frames where the maximum joint error per frame is below a specific threshold.

6.5 Discussion

Given the recent developments in Deep Learning, the creation of training data may now be the main bottleneck in practical applications of Machine Learning for hand pose estimation. Our method brings a much needed solution to the creation of accurate 3D annotations of hand poses. It avoids the need for motion capture systems, which are cumbersome and cannot always be used, and does not require complex camera setups. Moreover, it could also be applied to any other articulated structures, such as human bodies.

7

Leveraging Additional Synthetic Training Data[1]

Contents

In the previous chapter, we introduced a method to create labeled training data of articulated structures. Although this is an efficient and general purpose method, it still requires considerable amount of human labor especially when scaling the dataset to millions of images. However, a common side product of model-based hand annotation methods [271] is a synthetic rendering of the hand. In this chapter, we propose a simple and efficient method for exploiting synthetic images when training a Deep Network to predict a 3D pose from an image. The ability of using synthetic images for training a Deep Network is extremely valuable as it is easy to create a virtually infinite training set made of such images, while capturing and annotating real images can be very cumbersome. However, synthetic images do not resemble real images exactly, and using them for training can result in suboptimal performance. It was recently shown that for exemplar-based approaches, it is possible to learn a mapping from the exemplar representations of real images to the exemplar representations of synthetic images. Here, we show that this approach is more general and that a network can also be applied after the mapping to infer a 3D pose: At run-time, given a real image of the hand, we first compute the features for the image, map them to the feature space of synthetic images, and finally use the resulting features as input to another network that predicts the 3D pose. Since this network can be trained very effectively by

[1]The original publication [205], on which this chapter is based on, was done in collaboration with Mahdi Rad, who conducted experiments on 3D object pose estimation that were also presented in the original paper.

using synthetic images, it performs very well in practice, and inference is faster and more accurate than with an exemplar-based approach. We demonstrate our approach on the NYU dataset for 3D hand pose estimation from depth images. We show that it allows us to significantly outperform the state-of-the-art on this dataset. Our approach, DeepPrior++ with Feature Mapping, achieves state-of-the-art accuracy on the NYU dataset at the time of writing.

7.1 Introduction

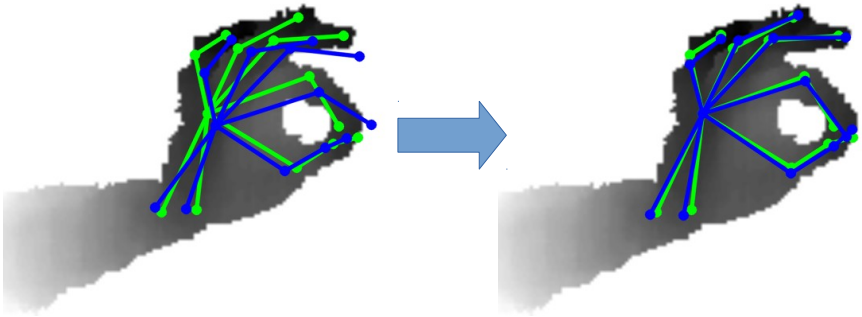


Figure 7.1: We propose a method for exploiting real and synthetic images to predict a 3D pose from a real image. This method allows us to outperform the state-of-the-art on the NYU dataset [271] for 3D hand pose estimation from depth images. **Left:** Estimated 3D pose using DeepPrior++, which we introduced in Section 4.2.5. **Right:** Estimated pose using the method proposed in this chapter. Green corresponds to ground truth, blue to our predictions. We obtain the best performances reported so far on this dataset.

The power of Deep Learning for inference from images has been clearly demonstrated over the past years, however, for many Computer Vision problems, inference is effective only if a large amount of training data is available. Typically this data is created and labeled manually, which is a task expensive in terms of both money and time. Compared to 2D problems where the labels can be directly defined in the training images, the problem is even exacerbated for 3D problems where the training images have to be labeled with 3D data. This 3D data cannot be guessed easily by the human annotator, and needs to be estimated with an *ad hoc* method, for example by using markers [94] or a semi-automatic approach [271].

Many works therefore aimed at using synthetic images created with computer graphics methods [3, 83, 141, 146, 244]. The resulting performances are usually suboptimal, as the synthetic images do not correspond exactly to real images. When some real images are available for training, which is often the case in practice, it is possible to use transfer learning [11, 171, 192, 219], where a first predictor is trained on real images and a second

one on synthetic images. By enforcing constraints on the parameters of the two predictors, the first predictor can benefit from a large amount of synthetic training images. Many works using Generative Adversarial Networks (GANs) have also been developed recently [24, 76, 230, 323], in which a first Deep Network is trained to generate real images and it competes with a second Deep Network trained to distinguish synthetic images from real ones.

However, transfer learning and *GANs* are two general approaches. While important by itself, the 3D pose estimation problem has some specificities that are not exploited by these two approaches. It was shown recently in [161] that, by synthesizing views of objects under the same pose as in some available real images, it is possible to learn a mapping between the features computed for a real image and the features computed in a synthetic image corresponding to the same pose. However, [161] applies this mapping to the descriptors of exemplars, which are matched using special layers computing a similarity score with reference exemplars. In this chapter, we show that this mapping can be used as input to a general network. We therefore train a network jointly with the feature mapping to predict the 3D pose of the hand from its synthetic images. We can use a virtually infinite number of training images to train this network, and it therefore performs very well. At run-time, given a real input image of the hand, we compute its image features, map them to the space of features of synthetic images, to finally predict the 3D pose of the hand from the mapped features.

As illustrated in Figure 7.1, we demonstrate our approach on the NYU dataset [271] for 3D hand pose estimation from depth images. Our experiments show that we can significantly outperform the state-of-the-art, by relying on our approach to exploit synthetic images. Moreover, pose inference is very efficient as it is performed by a Deep Network, in contrast to comparisons of exemplars as was done in [161].

In the remainder of this chapter, we discuss work related to using synthetic images for training Deep Networks, then present our approach and its evaluation on the NYU dataset.

7.2 Related Work on Using Synthetic Data for Training

A major problem in training Deep Networks is the acquisition of training data, but training data is critical for the success of Deep Networks [246]. An appealing solution is to use training samples rendered from 3D models [95, 120]. Such annotated samples are very easy to acquire, due to the presence of large scale 3D model datasets [35, 320]. However, using synthetic data requires special measures to prevent the network from overfitting on the synthetic appearance of the data. To prevent overfitting, [95] uses pretrained feature extractors from image classification networks, such as VGG [232], together with sophisticated data augmentation. While this is convenient as no real images are needed, it was only demonstrated on detection problems. [120] also uses synthetically generated images from 3D models with pretrained features, however, they require extensive refinement of

the initial network predictions, and we will show that by combining some real images and many synthetic images, we can reach better performances.
To improve the relevance of the synthetic images for training, [218] uses a small set of real images to estimate the rendering parameters that lead to image features similar to those of real images. However, this requires to parametrize the rendering process in such a way that it can be adapted from real images, which is restricted to *ad hoc* parameters, such as blurring, brightness, *etc.*
Other solutions of acquiring training data involve semi-supervised methods, such as the method introduced in Chapter 6 that propagates sparse annotations from a few keyframes to the full image sequences. However, it still requires labor intensive annotation of the keyframes.

Synthetic data on one side and real data on the other side can be seen as two different domains, which gives rise to domain adaptation methods. However, there can be significant differences between the synthetic and real images, which makes methods trained only on synthetic data perform poorly in practice [95]. Domain adaptation techniques provide a welcome solution to this problem, since it is easy to get training data in the synthetic domain, and it can be hard to acquire many training samples in the real domain.

For Deep Networks, fine-tuning is one of the most prominent and simple domain adaptation methods [75, 190]. This, however, can lead to severe overfitting, if there is only a small amount of training labels in the target domain available. Another way to handle the domain shift is to explicitly align the source and target distributions of the data. This can be achieved by quantifying the similarity of the two distributions and maximizing the similarity thereof. One popular metric is Maximum Mean Discrepancy (MMD) [80]. *MMD* can be either used to align the distributions of target and source features [102, 275], or by learning a transformation of the data such that the distributions match in a common subspace [11, 154, 171, 192]. [69] uses a deep feature extractor together with an additional classifier that predicts the domain for each sample. If the learned features are domain-invariant, such a classifier should exhibit poor performance. [275] adds an *MMD* loss to align the source and target data representations learned by Deep Networks.

However, [309] observed that feature transferability drops in higher layers of a Deep Network. To leverage this fact, [153] proposed a novel architecture that has the first few layers frozen, the mid layers fine-tuned, and the fully connected layers learned for each domain separately. The features of the fully connected layers are constrained by *MMD*. This works well for discriminative approaches that separate features into clusters, but not for regression problems. It also requires extensive task-specific validation on which layers to freeze, fine-tune, and transfer.

Very recently, [219] proposed a Siamese Network for domain adaptation, but instead of sharing the weights between the two streams, their method allows the weights to differ and only regularizes them to keep them related. This method is very general as it can learn adaptation between very different domains. We show in the results section that our approach outperforms [219], as we can exploit the specificities of our problem by rendering

synthetic images under the same poses as the real images to learn a mapping.

With the development of Deep Learning, *GANs* were also proposed for domain adaptation [76, 323], where a network is trained to transfer images from one domain to another domain. Although *GANs* are able to generate visually similar images in terms of appearance between different domains [24, 230], the synthesized images lack precision required to train 3D pose estimation methods. Especially for geometric tasks, such as 3D pose estimation, this shortcoming can be attributed to the lack of geometry in *GAN* models [308, 323]. An alternative to generating images in order to bridge the domain gap was presented by [274] and [62], who use a domain discriminator with adversarial loss to force a network to learn cross-domain features. Although this works well for discriminative applications, the features are not well suited for regression as we will discuss in the results section.

Another approach for domain adaptation is introduced in [29], which casts domain adaptation as an assignment problem, where samples from the target domain are assigned predefined classes from the source domain. This, however, strongly depends on the initialization of the features to obtain a semantically meaningful mapping, and for regression problems the undefined number of classes would make the problem intractable.

[161] proposed the method most related to ours, as they also learn a mapping from the real to synthetic domain. However, they consider an exemplar-based approach for 3D pose retrieval, and the mapping is applied to the exemplar representations. Relying on exemplars requires a discretization of the pose space in order to create the exemplars, and a nearest neighbor search. A fine discretization thus improves the accuracy of the estimated pose, but also slows down the nearest neighbor search. We show in this paper that it is possible to learn a similar mapping jointly with a network inferring the 3D pose after feature mapping. As our experiments show, our approach of retrieving the pose is thus both fast and accurate.

7.3 Feature Mapping for Using Synthetic Training Data

Since it is easy to create synthetic images, our goal is to exploit such images to guide learning, when training a Deep Network to predict a 3D pose from a real image. This real image can be a color image or a depth image, for example.

7.3.1 Training

We use synthetic images to train a feature extractor $f(\mathbf{x};\theta_f)$ together with a 3D pose predictor $h(\mathbf{f};\theta_h)$, which predicts a 3D pose given features $\mathbf{f}$ extracted from a given image $\mathbf{x}$. θ_f and θ_h denote the parameters for networks f and h, respectively. However, synthetic images do not resemble real images, and their features also vary significantly. Although this might not harm easier tasks such as object detection or object recognition, it is very important for accurate 3D pose prediction.

We therefore train a network $g(\mathbf{f};\theta_g)$ with parameters θ_g to map features of the real

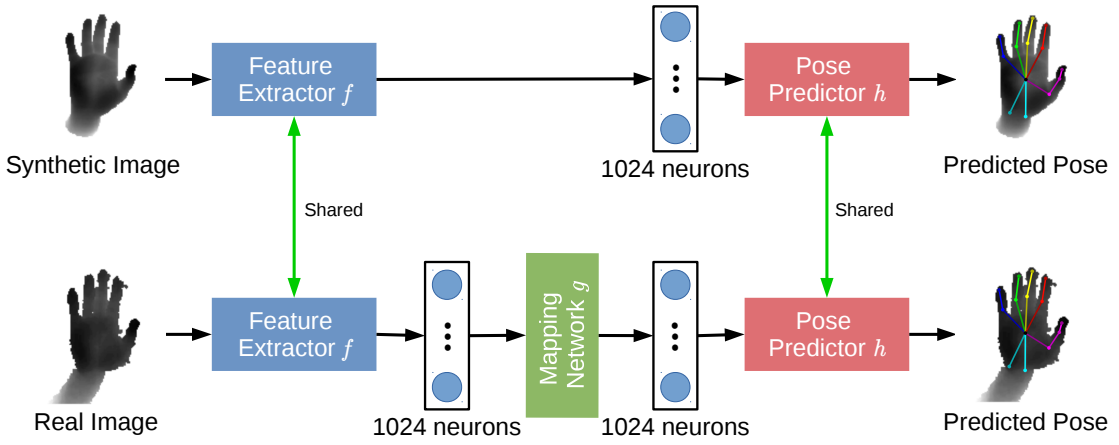


Figure 7.2: The overall model architecture that we use for training is made of two streams, one for the synthetic and one for the real images. Each stream contains a feature extractor f (blue) and a pose predictor h (red). The real stream contains the feature mapping network g (green) to map features extracted from a real image to the feature space of synthetic images. The parameters of the feature extractor f are shared between the two streams, as well as the parameters of the pose predictor h. We apply g only to the features from real images, not on the features from synthetic images. We can thus train the pose predictor on synthetic images, and apply it to real images at run-time without being affected by the domain gap.

images into the synthetic feature space, before they are used as input to the h network that predicts the 3D pose from image features. Figure 7.2 shows the three networks and how they are connected. Specifically, we implement the networks in a two stream architecture. One stream takes the real images and the other stream takes the synthetic images. The parameters of the two streams are shared, except for the mapping network g.

Network g is trained using pairs of images, each pair is made of one of the available real images and of one synthetic image of the hand rendered under the same 3D pose as in the real image. During training, we minimize the distance between the features extracted from the synthetic images and the features extracted from the real image after mapping by g. For 3D hand pose estimation from depth images, we directly generate a depth image of the 3D model under the same pose. The first row of Figure 7.3 shows an example.

More exactly, we train the three networks f, h, and g jointly on the training set $\mathcal{T} = \mathcal{T}^{\mathcal{S}} \cup \mathcal{T}^{\mathcal{R}}$ where $\mathcal{T}^{\mathcal{S}} = \{(\mathbf{x}_i^{\mathcal{S}}, \mathbf{y}_i^{\mathcal{S}})\}_i$ denotes a training set of synthetic images and their corresponding 3D labels, and $\mathcal{T}^{\mathcal{R}} = \{(\mathbf{x}_i^{\mathcal{R}}, \mathbf{y}_i^{\mathcal{R}})\}_i$ is made of real images and their 3D labels. We also use a training set $\mathcal{T}^{\mathcal{M}}$ made of the real images $\mathbf{x}_i^{\mathcal{R}}$ in $\mathcal{T}^{\mathcal{R}}$, each paired to a synthetic image generated under the same pose as mentioned above. Jointly training f, h, and g helps learning to extract image features such that they are transferable from the real domain to the synthetic domain.

We optimize the following loss function over the parameters of the three networks:

$$\mathcal{L}(\theta_f, \theta_h, \theta_g; \mathcal{T}^{\mathcal{S}}, \mathcal{T}^{\mathcal{R}}, \mathcal{T}^{\mathcal{M}}) = \mathcal{L}_{h_S} + \beta \mathcal{L}_{h_R} + \gamma \mathcal{L}_g \,. \tag{7.1}$$

β and γ are meta-parameters to control the interaction of the losses.

$\mathcal{L}_{h_S}$ is the loss for predicting the poses for synthetic images:

$$\mathcal{L}_{h_S} = \sum_{(\mathbf{x}_s, \mathbf{y}_s) \in \mathcal{T}^{\mathcal{S}}} \|h(f(\mathbf{x}_s; \theta_f); \theta_h) - \mathbf{y}_s\|^2 . \tag{7.2}$$

Note that we compose network f that extracts image features from image $\mathbf{x}$ and network h that predicts a 3D pose from these features.

In practice, the 3D pose $\mathbf{y}$ is simply parameterized as a vector made of the 3D locations of the hand joints normalized as in Chapter 4, which allows us to use the Euclidean norm in Eqn. (7.2).

$\mathcal{L}_{h_R}$ is a loss function equivalent to $\mathcal{L}_{h_S}$ but for the real images:

$$\mathcal{L}_{h_R} = \sum_{(\mathbf{x}_r, \mathbf{y}_r) \in \mathcal{T}^{\mathcal{R}}} \|h(g(f(\mathbf{x}_r; \theta_f); \theta_g); \theta_h) - \mathbf{y}_r\|^2 , \tag{7.3}$$

where we compose f, g, and h together, to first extract image features, then map them to the space of image features for synthetic images, and finally predict the 3D pose from these mapped features.

$\mathcal{L}_g$ is the loss to learn the mapping between the features extracted from the real images to the features extracted from the synthetic images:

$$\mathcal{L}_g = \sum_{(\mathbf{x}_r, \mathbf{x}_s) \in \mathcal{T}^{\mathcal{M}}} \|g(f(\mathbf{x}_r; \theta_f); \theta_g) - f(\mathbf{x}_s; \theta_f)\|^2 . \tag{7.4}$$

7.3.2 Effect of the Learned Mapping

To understand better the effect of the learned mapping, we computed the distributions of the absolute differences between the synthetic feature vectors and real feature vectors, as computed by network f, before and after mapping of the real feature vectors by network g. The distributions remain surprisingly close, as their means and standard deviations are $(\mu_1 = 1.60, \sigma_1 = 1.64)$ and $(\mu_2 = 1.30, \sigma_2 = 1.50)$ respectively.

However, considering distributions can only provide a limited view. To get a finer insight, we took a pair of real and synthetic images under the same pose, and we plotted the absolute differences between the coefficients of their feature vectors, first without mapping, then after mapping of the real feature vector by network g. The differences are shown in Figure 7.3(c) and Figure 7.3(d). Blue corresponds to small differences, and red to large differences. It appears that the mapping mostly removes the large differences without changing the smaller differences. We repeated this experiment on other image pairs and observed a similar behavior. Our interpretation is that for each pair, only a few feature coefficients are responsible for the domain gap, and they can be attenuated by the mapping.

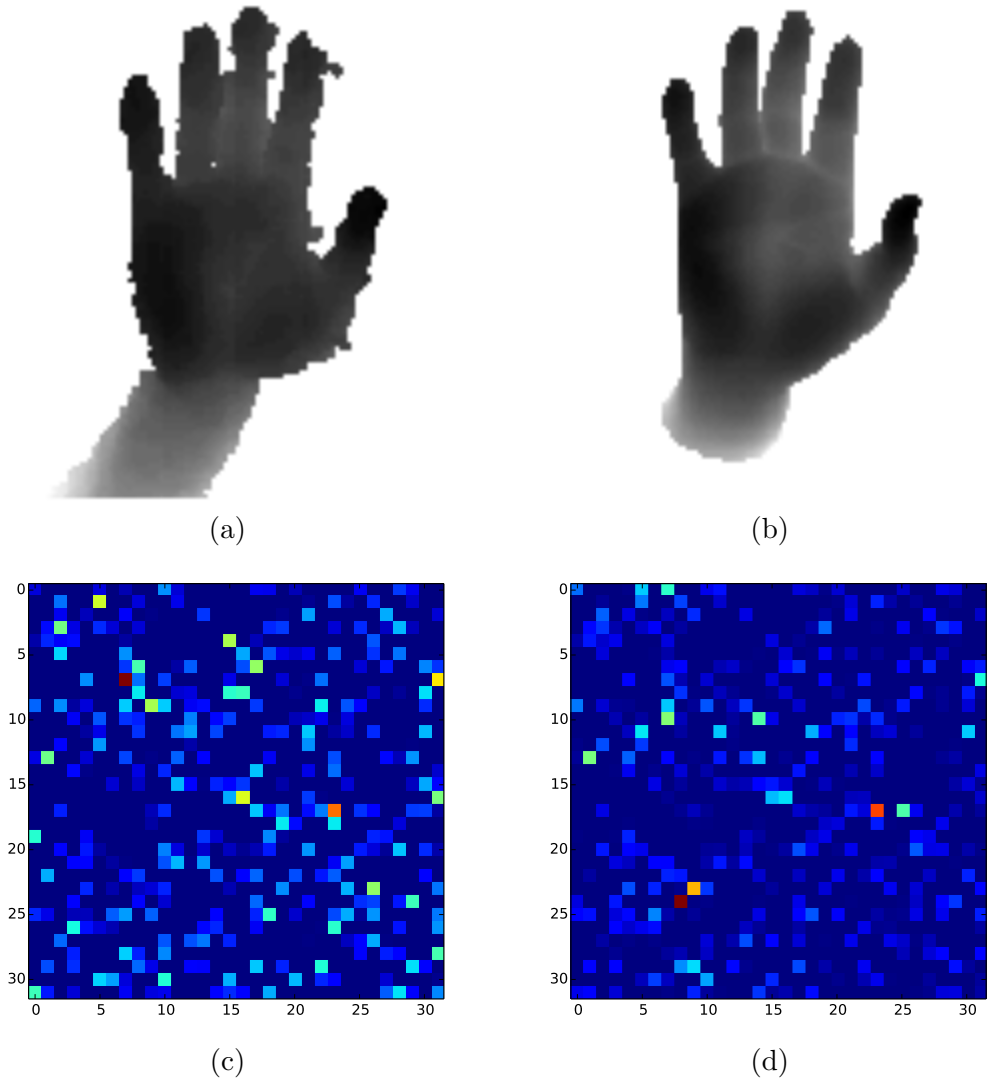


Figure 7.3: The effect of the mapping learned by network g using a real image (a) and the corresponding synthetic image (b). The second row shows the absolute differences between the synthetic and real feature vectors (reshaped to images for better visualization) before (c) and after mapping (d). The mapping mostly removes the large differences.

7.3.3 Pose Prediction

At run-time, given a real image $\mathbf{x}$, we predict the corresponding 3D pose $\mathbf{y}$ by composing the three networks together:

$$\mathbf{y} \leftarrow h(g(f(\mathbf{x}; \widehat{\theta_f}); \widehat{\theta_g}); \widehat{\theta_h}) \ ,$$

where $(\widehat{\theta_f}, \widehat{\theta_g}, \widehat{\theta_h})$ are the networks' parameters found during training. Note that this composition of the three networks can be implemented as a single network to improve efficiency.

7.3.4 Network Details

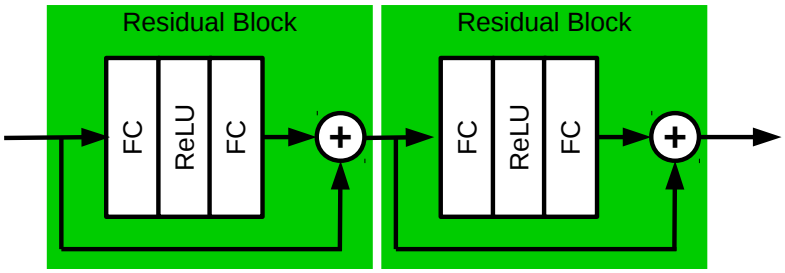


Figure 7.4: Our model architecture of the mapping network g used to map features extracted from a real image to the feature space of synthetic images. It consists of two Residual blocks [92]. FC denotes a fully connected layer, and ReLU a rectified linear unit.

As shown in Figure 7.4, we use two Residual blocks [92] for network g to map feature vectors of size 1024. Each fully connected layer within the Residual block has 1024 neurons.

We use an architecture similar to the one of [173, 181] for the feature extraction network f: It is similar to the 50-layer Residual Network [92] with four residual modules. We remove the Global Average Pooling [92], which we experienced during our experiments significantly reduces localization accuracy of the joints, and add two fully connected layers with 1024 neurons each. f is trained from scratch. We use a single fully connected layer with 42 outputs—3 for each of the 14 joints—for the pose prediction network h, together with a bottleneck layer as introduced in Chapter 4.

For the parameters of the loss function in Eqn. (7.1), we use $\beta = 1$, which gives the same weight to the synthetic and real samples, and $\gamma = 0.2$, which gives a good trade-off between pose loss and feature mapping loss. We optimize the loss using gradient descent, specifically the ADAM algorithm [125] with a learning rate of 10^{-4} and a batch size of 64. In order to improve convergence, we pretraining the networks with synthetic data only, which in practice gives better results compared to starting from a random initialization.

7.4 Evaluation

In this section, we present and discuss the results of our evaluation for 3D hand pose estimation from depth images.

7.4.1 Training Set Creation

We evaluate it on 3D hand pose estimation from single view depth images. For this experiment, we consider the NYU dataset [271]. We follow the protocol of [271] and predict a subset of 14 joints. We use the pipeline described in Section 4.2.5 to preprocess the depth images: It crops a 128×128 patch around the hand location, and normalizes its depth values to the range of $[-1, +1]$.

Creating synthetic depth images for hand poses is a relatively simple problem. In practice, we use the 3D hand model of [271] to render synthetic views of a hand. However, it should be noted that the noise present in real depth images captured with a structured-light sensor is difficult to simulate, and our synthetic depth images do not contain any noise. We use 5M synthetic images of the hand that are rendered online during training from poses of the training set perturbed with randomly added articulations.

(a) Synthetic depth images. **(b)** Real depth images.

Figure 7.5: Importance of noise in real depth images. (a): Synthetic images; (b): Real images for the same poses as in (a). There is a significant amount of sensor noise present in the real depth images, which considerably deteriorates the results when using synthetic training data only, as we show below.

7.4.2 Comparison with Transfer Learning

We first compare our method to other domain adaptation methods: The transfer learning method proposed in [219], the GRL method of [69], and the ADDA method of [274]. We also compare to the DDC method [275], designed for classification. We therefore changed the classification loss to our regression loss on the 3D pose. A comparison to these other transfer learning methods is shown in Table 7.1. Although the other transfer learning methods improve the accuracy compared to training with real data only, the results obtained by our proposed Feature Mapping are much more accurate.

We also tried the method of [274], which performed actually more poorly on this problem. This can be explained by the fact that this method does not guarantee that the learned features carry enough information for predicting a pose, as this method was designed for classification, not regression.

Method	[219]	GRL [69]	DDC [275]	DeepPrior++	DeepPrior++ w/ Feature Mapping
Avg. 3D error	9.8 mm	9.4 mm	11.1 mm	12.3 mm	**7.4 mm**

Table 7.1: Comparison of different domain adaptation methods ([219], GRL [69], and DDC [275]) on 3D hand pose estimation using the NYU dataset [271]. All methods predict the 3D joint locations using regression and we compare them using the average Euclidean 3D error. DeepPrior++ denotes the results obtained using only the real images. Our method, DeepPrior++ with Feature Mapping, outperforms the other domain adaptation methods on this problem.

7.4.3 Comparison with Baselines

We compare our method to very recent state-of-the-art methods: *DeepPrior++* introduced in Section 4.2.5 that integrates a prior on the 3D hand poses into a Deep Network; *REN* [82] relies on an ensemble of Deep Networks, each operating on a region of the input image; *Lie-X* [300] uses a sophisticated tracking algorithm constrained to the Lie group; *Crossing Nets* [283] uses an adversarial training architecture; Neverova *et al.* [178] proposed a semi-supervised approach that incorporates a semantic segmentation of the hand; *DeepModel* [321] integrates a 3D hand model into a Deep Network; *DISCO* [22] learns the posterior distribution of hand poses; *Feedback* introduced in Chapter 5 that uses an additional Deep Network to improve results of an initial prediction; *Hand3D* [52] uses a volumetric Convolutional Neural Network (CNN) to process a point cloud.

Table 7.2 compares the different methods we consider using the average Euclidean distance between ground truth and predicted joint 3D locations, which is a *de facto* standard for this problem. When training on synthetic data only, the error on the real test images is 21 mm, which suggests that the network severely overfits to the synthetic depth images and cannot generalizes to real depth images, which are often very noisy. By using DeepPrior++ with Feature Mapping, we achieve an error of 7.4 mm, which improves the state-of-the-art by 4.9 mm or almost 40%. DeepPrior++ with Feature Mapping achieves state-of-the-art accuracy at the time of writing, and when using ground truth detection, the accuracy can be improved to an average 3D error of 5.5 mm.

Figure 7.7 shows the fraction of frames where all joints of a frame are within a maximum distance from the ground truth. This is a very difficult metric since a single erroneous joint can deteriorate the result of a frame [183, 260]. We significantly outperform all previous works on this difficult metric, by almost 40% at an error threshold of 20 mm.

Figure 7.6 shows some qualitative results. When only synthetic depth images are used, the predictions for synthetic depth images are typically very good, but the predictions for real images are bad. When using our method, the predicted poses significantly improve, especially in the presence of noise in the depth images. In Figure 7.8 we qualitatively compare the 3D hand pose estimates obtained with our approach to those obtained with DeepPrior++.

Method	Average 3D error
Neverova *et al.* [178]	14.9 mm
Crossing Nets [283]	15.5 mm
Lie-X [300]	14.5 mm
REN [82]	13.4 mm
DeepPrior++ (Section 4.2.5)	12.3 mm
Feedback (Chapter 5)	16.2 mm
Hand3D [52]	17.6 mm
DISCO [22]	20.7 mm
DeepModel [321]	16.9 mm
DeepPrior++ synthetic only	21.1 mm
DeepPrior++ w/ Feature Mapping	**7.4 mm**

Table 7.2: Quantitative evaluation on the NYU dataset [271]. We compare the average Euclidean 3D error of the predicted poses with state-of-the-art methods on the NYU dataset. The numbers are reported for the real test images.

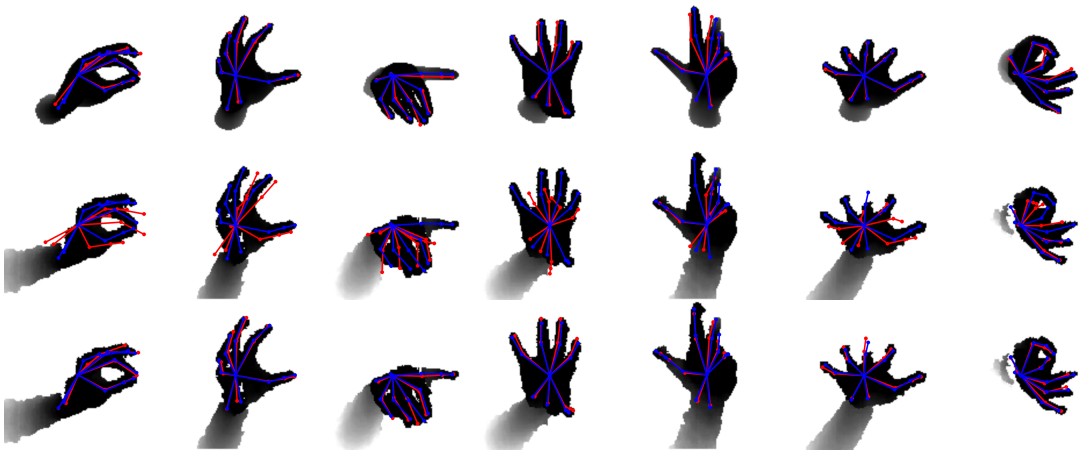


Figure 7.6: Qualitative results on the NYU dataset [271]. **Top row**: Predicted poses on synthetic test depth images. **Middle row**: Predicted poses on real test depth images, when training on synthetic depth images only. Training only on synthetic depth images does not generalize well to real depth images. **Bottom row**: Predicted poses when using our proposed DeepPrior++ with Feature Mapping. We plot the 2D projection of the 3D pose prediction. Blue denotes ground truth annotations, red are our predictions.

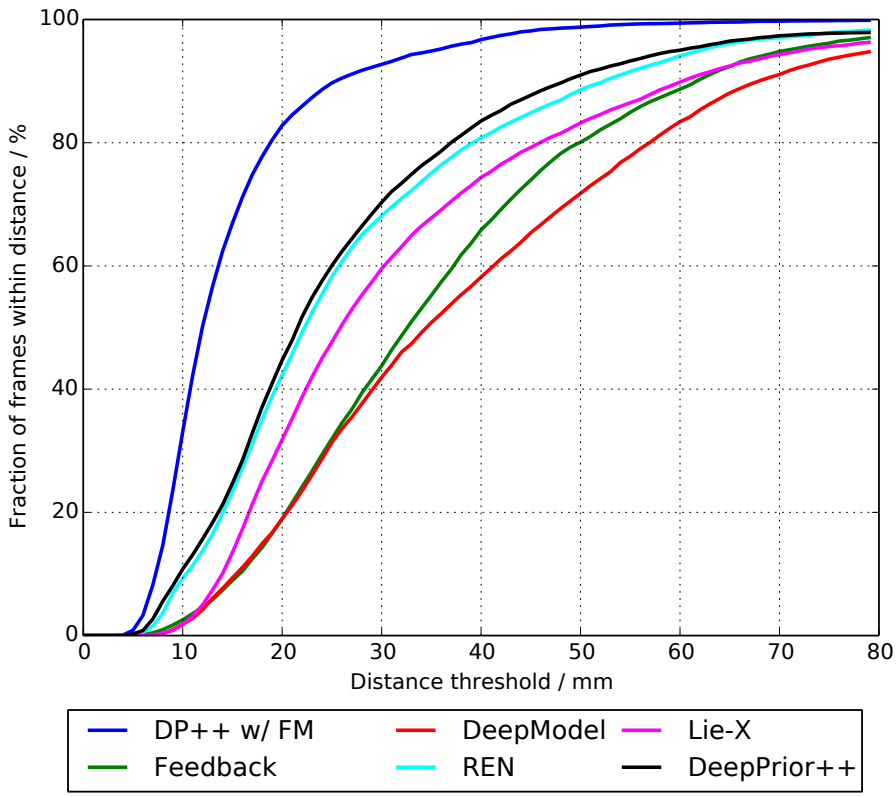


Figure 7.7: Comparison with state-of-the-art methods on the NYU dataset [271]. We plot the fraction of frames where all joints of a frame are within a maximum distance from the ground truth. A larger area under the curve indicates better results. The proposed DeepPrior++ with Feature Mapping (DP++ w/ FM) performs best among all other methods.

Further, we compare against DeepPrior++, which is the second best performing method identified. Figure 7.9 shows a comparison with this method on the NYU dataset [271], where we compare the distributions of the average 3D joint errors. DeepPrior++ with Feature Mapping outperform the original DeepPrior++ on almost all frames.

7.4.4 Influence of the Number of Real Training Images

Capturing real data is very cumbersome and time consuming, which motivates the use of synthetic data. We therefore evaluate the influence of the number of real training images on the NYU dataset [271]. The dataset contains 72k real images that we randomly subsample to reduce the number of real training images, while keeping the number of synthetic images the same. Figure 7.10 shows the influence of the number of real training images on the final results. Using our method systematically improves the results compared to using only real images for training. DeepPrior++ with Feature Mapping requires less than 1%

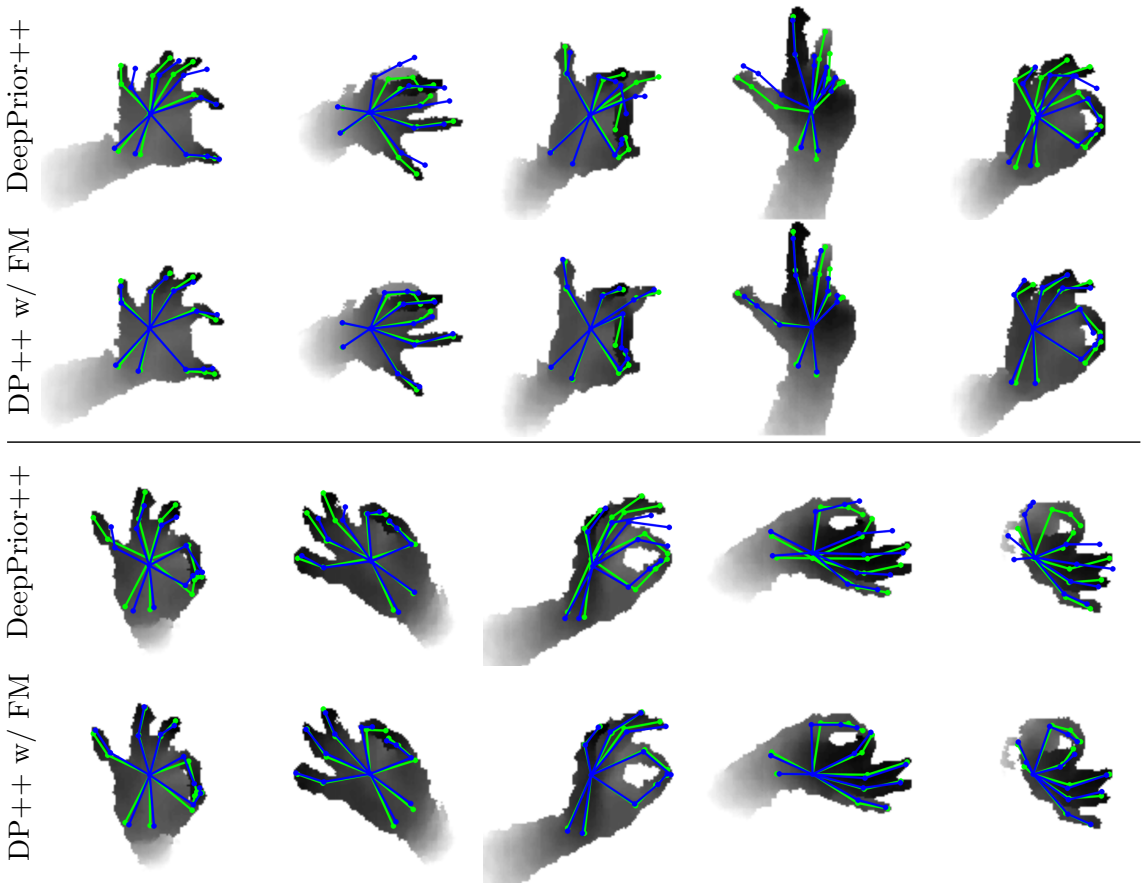


Figure 7.8: **Top rows:** Estimated 3D poses for the hand on the NYU dataset [271] by using DeepPrior++. **Bottom rows:** Estimated poses using DeepPrior++ with Feature Mapping (DP++ w/ FM). Green corresponds to ground truth, blue to our predictions.

of the amount of training images and still has higher accuracy compared to the original DeepPrior++ trained only on real images. We attribute this to the highly correlated poses in the training data, which allows to omit many training samples without loosing much accuracy and the generalization of the feature mapping network.

7.4.5 Runtime

We implemented our approach in Theano [15] and the code runs on an Intel Core i7 with an nVidia Titan GPU. Our approach takes 8.6 ms for the pose inference including feature mapping, which requires only a few matrix multiplications in addition to the feature extraction. This corresponds to more than 110 frames per second (fps). These computation times should be compared to those of the exemplar-based approach of [161], which reports runtimes of several seconds per image.

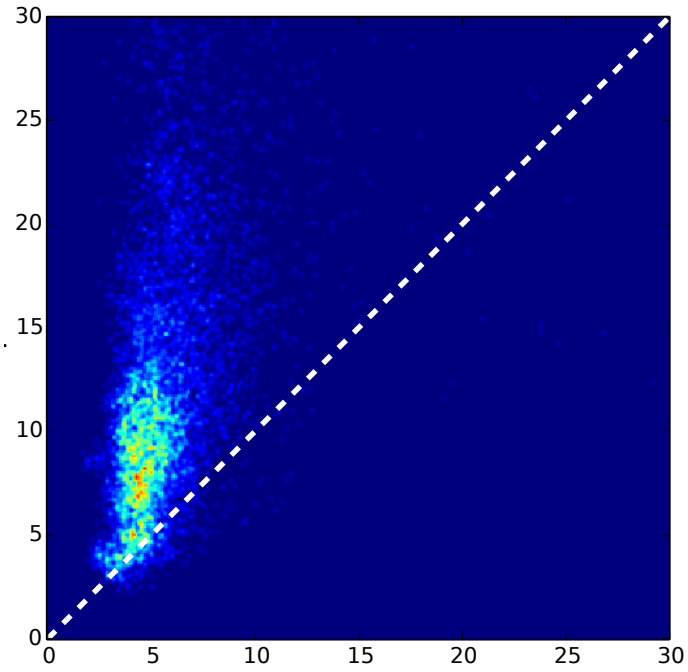


Figure 7.9: Joint distribution of the average Euclidean 3D distance (in mm) on the NYU dataset [271] for DeepPrior++ with Feature Mapping (on the x-axis) and the original DeepPrior++ (on the y-axis). The fact that the distribution mass is over the main diagonal shows that Feature Mapping improves the pose estimates for most of the images. The improvement is often very large on the NYU dataset.

7.5 Discussion

We showed that domain transfer between synthetic and real images can be achieved easily in the case of 3D pose inference. We presented an approach that learns a mapping between the two domains from synthetic images rendered under the same pose as the available real training images, jointly with feature extraction and pose inference. Our method is simple to implement, can be easily optimized, is very efficient at run-time, and allowed us to outperform the state-of-the-art on a popular dataset for 3D hand pose estimation.

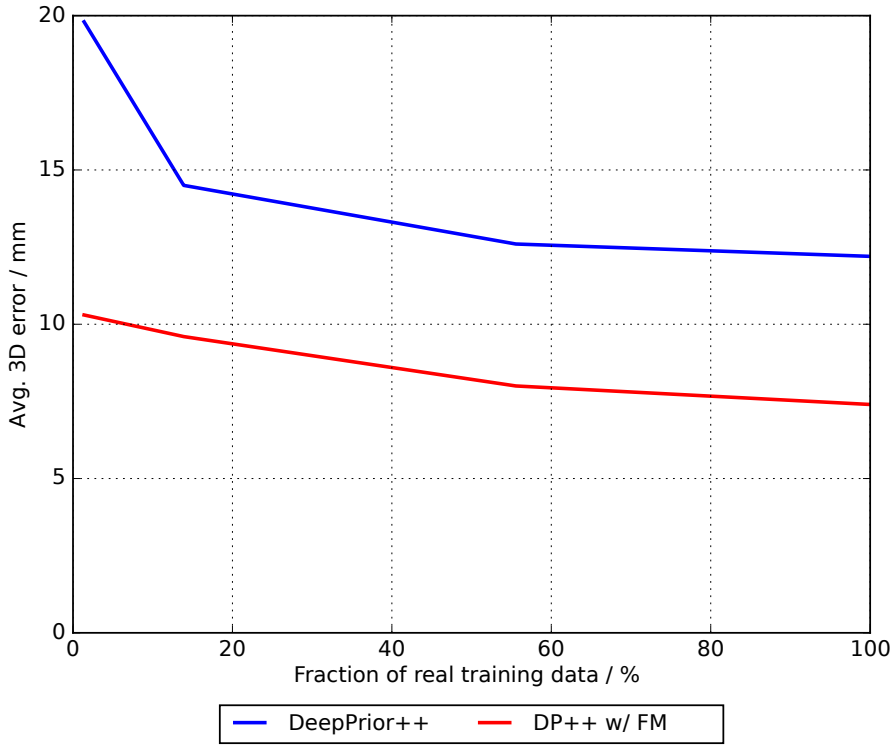


Figure 7.10: Influence of the number of real images for 3D hand pose estimation on the NYU dataset [271]. We report the average 3D error. DeepPrior++ corresponds to the results obtained when only real images are used for training. When using the same number of real training images, DeepPrior++ with Feature Mapping (DP++ w/ FM) systematically improves the performance.

8

3D Pose Estimation from Color Images without Labeled Color Images[1]

Contents

In the previous chapter, we showed how to exploit additional synthetic data. In case of 3D hand pose estimation, generating synthetic depth images is a rather simple task, but recent work in this field [172, 237, 325] tends to use color images as input. However, rendering color images for hands is much more difficult than rendering depth images, since it would need to consider lighting and texture as well. Therefore, we introduce a novel learning method for 3D pose estimation from color images. While acquiring annotations for color images is a difficult task, our approach circumvents this problem by learning a mapping from paired color and depth images captured with an RGB-D camera. We jointly learn the 3D pose from synthetic depth images that are easy to generate, and learn to align these synthetic depth images with the real depth images. We show our approach for the task of 3D hand pose estimation from color images only. Our method achieves performances comparable to state-of-the-art methods on a popular benchmark dataset, without requiring any labels for the color images.

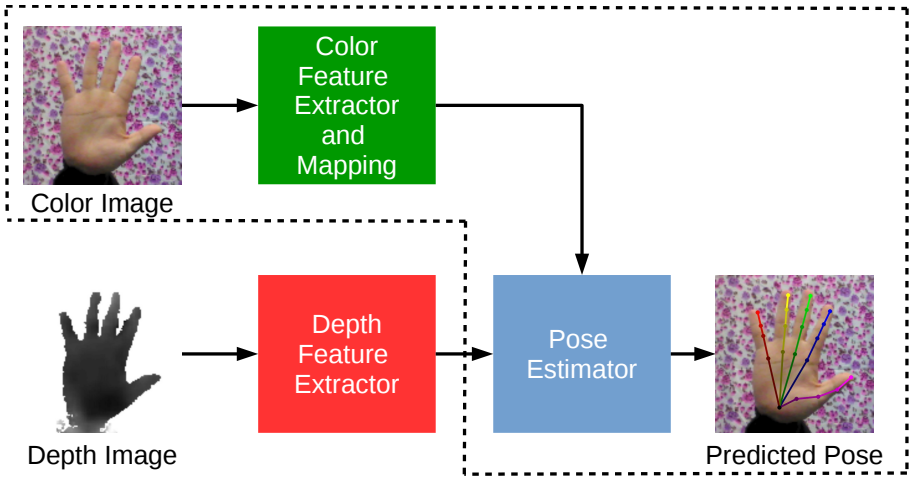


Figure 8.1: Method overview. We train a depth feature extractor (red box) together with a pose estimator (blue box) on synthetic data. We also train a second network (green box), which extracts image color features and maps them to the depth space, given color images and their corresponding depth images. At run-time, given a color image, we map color features to depth space in order to use the pose estimator to predict the 3D pose of the hand (dashed lines). This removes the need for labeled color images.

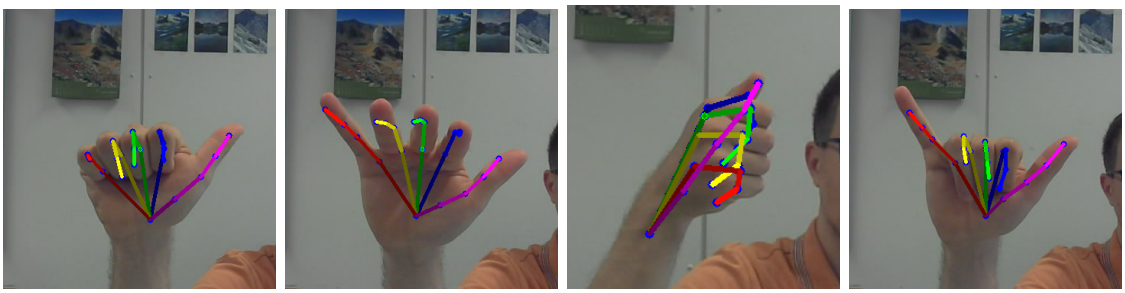


Figure 8.2: Our method allows accurate 3D pose estimation from color images without labeled color images. We show the estimated 3D joint locations projected to the color image.

8.1 Introduction

As already discussed in the previous chapters, Deep Learning methods require a large amount of training data to perform well [181, 203, 205, 264, 297]. To create training data, the labeling is usually done with the help of markers [88, 311], which is very cumbersome, expensive, or sometimes even impossible, especially from color images. For example, markers cannot be used for 3D hand labeling of color images, as they change the appearance of the hand.

[1]The original publication [204], on which this chapter is based on, was done in collaboration with Mahdi Rad, who conducted experiments on 3D object pose estimation that were also presented in the original paper.

Another direction is to use synthetic images for training. However, synthetic images do not exactly look like real images. Generative Adversarial Networks (GANs) [24, 76, 230, 323] or transfer learning techniques [69, 171, 192, 219], as the method we introduced in Chapter 7, can be used to bridge the domain gap between real and synthetic images. However, these approaches still require some annotated real images to learn the domain transfer. In Chapter 7 we relied on registered real images to compute a direct mapping between the image features of real and synthetic images, but it also requires some labeled real images.

In this chapter, we propose a method that learns to predict a 3D pose from color images, without requiring labeled color images. Instead, it exploits labeled depth images. These depth images can be real depth images, which are easier to label than color images, and are already readily available for some problems. More interestingly, they can also be synthetic depth images: Compared to color images, synthetic depth images are easier to render, as there is no texture or illumination present in these images.

An overview of our approach is shown in Figure 8.1. Our main idea is to bridge the domain gap between color images and these synthetic depth images in two steps, each one solving an easier problem than the original one. We use an RGB-D camera to capture a set of pairs made of color and depth images that correspond to the same view. Capturing such a set can be done by simply moving the camera around. We apply Feature Mapping, which we introduced in Chapter 7, to this set and learn to map the features from the color images to corresponding depth images. However, this mapping alone is not sufficient: A domain gap between the depth images captured by the RGB-D camera and the available labeled depth images remains, since the labeled depth images could be captured with another RGB-D camera or rendered synthetically. Fortunately, this remaining gap is easier to bridge than the domain gap between real and synthetic color images, since illumination and texture effects are not present in depth images. To handle it, we use Maximum Mean Discrepancy (MMD) [80] to measure and minimize the distance between the means of the features of the real and synthetic depth images mapped into a Reproducing Kernel Hilbert Space (RKHS). *MMD* is a popular in domain transfer method [219] since it does not require correspondences to align the features of different domains and can be efficiently implemented.

Our approach is general. It can be applied to many other applications, such as 3D object pose estimation, human pose estimation, etc. Furthermore, in contrast to color rendering, no prior information about object's texture has to be known. Figure 8.2 shows results on 3D hand pose estimation from color images on the STB dataset [316]. Our method achieves performance comparable to state-of-the-art methods on this dataset without requiring any labels for the color images.

In the remainder of this chapter, we discuss related work, then present our approach and its evaluation.

8.2 Related Work on Domain Transfer Learning

As we mentioned in the introduction, it is difficult to acquire annotations for real training data, and training on synthetic data leads to poor results [120, 203]. This is an indication for a domain gap between synthetic and real training data. Moreover, using synthetic data still requires accurately textured models [32, 95, 120, 203] that require large amount of engineering to model. On the other hand, synthetic depth data is much simpler to produce, but still it requires a method for domain transfer.

A popular method is to align the distributions for the extracted features from the different domains. *GANs* [76] and Variational Autoencoders (VAEs) [126] can be used to learn a common embedding for the different domains. This usually involves learning a transformation of the data such that the distributions match in a common subspace [69, 171, 192]. [237] learns a shared embedding of images and 3D poses, but it requires annotations for the images to learn this mapping. Although *GANs* are able to generate visually similar images between different domains [323], the synthesized images lack precision required to train 3D pose estimation methods [24, 205]. Therefore, [172] developed a sophisticated *GAN* approach to adapt the visual appearance of synthetically rendered images to real images, but this still requires renderings of high-quality synthetic color images.

To bridge this domain gap, we introduced a method in Chapter 7 that predicts synthetic features from real features and uses these predicted features for inference, but this works only for a single modality, *i.e.*, depth or color images, and requires annotations from both domains. Similarly, [85] transfers supervision between images from different modalities by learning representations from a large labeled modality as a supervisory signal for training representations for a new unlabeled paired modality. In our case, however, we have an additional domain gap between real and synthetic depth data, which is not considered in their work. Also, [31] aims at transforming the source features into the space of target features by optimizing an adversarial loss. However, they have only demonstrated this for classification and this approach works poorly for regression, as shown in Chapter 7. [219] proposed a Siamese Network for domain adaptation, but instead of sharing the weights between the two streams, their method allows the weights to differ and only regularizes them to keep them related. However, we have shown in Chapter 7 that the adapted features are not accurate enough for 3D pose estimation.

Differently, [312] transfers real depth images to clean synthetic-looking depth images. However, this requires extensive hand-crafted depth image augmentation to create artificial real-looking depth images during training, and modeling the noise of real depth cameras is difficult in practice. [267] proposed to randomize the appearance of objects and rendering parameters during training, in order to improve generalization of the trained model to real-world scenarios. However, this requires significant engineering effort and there is no guarantee that these randomized renderings cover all the visual appearances of a real-world scene.

Several works [97, 236] proposed a fusion of features from different domains. [97] fuses

color and depth features by using labeled depth images for a few categories and adapts a color object detector for a new category such that it can use depth images in addition to color images at test time. [236] proposed a combination method that selects discriminative combinations of layers from the different source models and target modalities and sums the features before feeding them to a classifier. However, both works require annotated images in all domains. [103] uses a shared network that utilizes one modality in both source and target domain as a bridge between the two modalities, and an additional network that preserves the cross-modal semantic correlation in the target domain. However, they require annotations in both domains, whereas we only require annotations in one, *i.e.*, the synthetic, domain that are much easier to acquire.

When comparing our work to these related works on domain transfer learning, these methods either require annotated examples in the target domain [31, 97, 103, 172, 205, 236], are restricted to two domains [31, 83, 97, 205, 236], or require significant engineering [267, 312]. By contrast, our method does not require any annotations in the target domain, *i.e.*, color images, and can be only trained on synthetically rendered depth images that are easy to generate, and the domain transfer is trained from real data that can be easily acquired using a commodity RGB-D camera.

8.3 Domain Transfer for 3D Pose Estimation from Color Images

Given a 3D model of a hand, it is easy to generate a training set made of many depth images of the hand under different 3D poses. Alternatively, we can use an existing dataset of labeled depth images. We use this training set to train a first network to extract features from such depth images, and a second network, the *pose estimator*, to predict the 3D pose of a hand, given the features extracted from a depth image. Because it is trained on many images, the pose estimator performs well, but only on depth images.

To apply the pose estimator network to color images, we train a network to map features extracted from color images to features extracted from depth images, as was done in Chapter 7 between real and synthetic images. To learn this mapping, we capture a set of pairs of color and depth images that correspond to the same view, using an RGB-D camera. In order to handle the domain gap between the real and synthetic depth images of two training sets, we apply *MMD* [80], which aims to map features of each training set to a *RKHS* and then minimizes the distance between the means of the mapped features of the two training sets. An overview of the proposed method is shown in Figure 8.3.

8.3.1 Learning the Mapping

More formally, let $\mathcal{T}^{\mathcal{S}} = \{(\mathbf{x}_i^{\mathcal{S}}, \mathbf{y}_i)\}_i$ be a training set of synthetically rendered depth images $\mathbf{x}_i^{\mathcal{S}}$ using a 3D rendering engine under 3D poses $\mathbf{y}_i$. A second training set $\mathcal{T}^{\texttt{RGB-D}} = \{(\mathbf{x}_i^{\mathcal{R}}, \mathbf{x}_i^{\mathcal{D}})\}_i$ consists of pairs of color images $\mathbf{x}_i^{\mathcal{R}}$, and their corresponding depth images

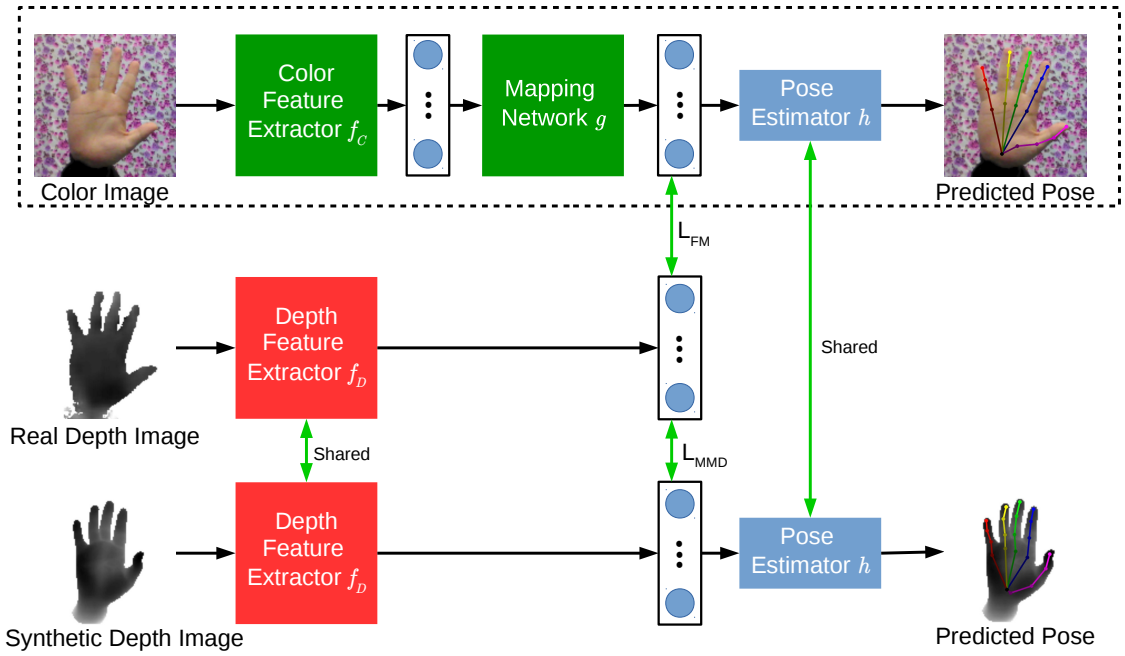


Figure 8.3: Detailed overview of our approach. It consists of three data streams, one for each domain. The lower two streams take depth images as input, *i.e.*, synthetic and real depth images, respectively, and extract features using the network $f_{\mathcal{D}}$. The upper stream takes color images as input and uses the mapping network g to map the color features to the depth features used for pose prediction with network $h_{\mathcal{D}}$. The parameters of the depth feature extractor $f_{\mathcal{D}}$ and pose predictor $h_{\mathcal{D}}$ are shared between the synthetic and real depth image (green arrows). Between the synthetic and the real depth feature we have the *MMD* loss $\mathcal{L}_{MMD}$ and between the real color and real depth features we use the feature mapping loss $\mathcal{L}_{FM}$. For inference at test time, we use only the upper stream within the dashed lines that takes a real color image as input, extracts features using the network $f_{\mathcal{C}}$, maps these features to the depth space using g, and uses the pose estimator $h_{\mathcal{D}}$ to predict the 3D pose.

$\mathbf{x}_i^{\mathcal{D}}$. We jointly train four networks: the feature extractor for depth images $f_{\mathcal{D}}$, the feature extractor for color images $f_{\mathcal{C}}$, the pose estimator $h_{\mathcal{D}}$, and the feature mapping network g, on the training sets $\mathcal{T}^{\mathcal{D}}$ and $\mathcal{T}^{\texttt{RGB-D}}$.

We optimize the following loss function over the parameters of networks $f_{\mathcal{D}}$, $h_{\mathcal{D}}$, $f_{\mathcal{C}}$, and g as:

$$\begin{aligned}&\mathcal{L}(\theta_D, \theta_h, \theta_C, \theta_g; \mathcal{T}^{\mathcal{S}}, \mathcal{T}^{\texttt{RGB-D}}) = \\ &\mathcal{L}_P(\theta_D, \theta_h; \mathcal{T}^{\mathcal{S}}) + \beta \mathcal{L}_{FM}(\theta_D, \theta_C, \theta_g; \mathcal{T}^{\texttt{RGB-D}}) + \gamma \mathcal{L}_{MMD}(\theta_D; \mathcal{T}^{\mathcal{S}}, \mathcal{T}^{\texttt{RGB-D}}) \,,\end{aligned} \tag{8.1}$$

where θ_D, θ_h, θ_C, and θ_g are the parameters of networks $f_{\mathcal{D}}$, $h_{\mathcal{D}}$, $f_{\mathcal{C}}$, and g, respectively. The losses $\mathcal{L}_P$ for the pose, $\mathcal{L}_{FM}$ for the feature mapping between color and depth features, and $\mathcal{L}_{MMD}$ for the *MMD* between synthetic and real depth images are weighted by meta parameters β and γ.

$\mathcal{L}_P$ is the sum of the errors for poses predicted from depth images:

$$\mathcal{L}_P(\theta_D, \theta_h; \mathcal{T}^{\mathcal{S}}) = \sum_{(\mathbf{x}_i^{\mathcal{S}}, \mathbf{y}_i) \in \mathcal{T}^{\mathcal{S}}} \| h_{\mathcal{D}}(f_{\mathcal{D}}(\mathbf{x}_i^{\mathcal{S}}; \theta_D); \theta_h) - \mathbf{y}_i \|^2 . \tag{8.2}$$

$\mathcal{L}_{FM}$ is the loss used to learn to map features extracted from depth images to features extracted from their corresponding color images:

$$\mathcal{L}_{FM}(\theta_D, \theta_C, \theta_g; \mathcal{T}^{\texttt{RGB-D}}) = \sum_{(\mathbf{x}_i^{\mathcal{R}}, \mathbf{x}_i^{\mathcal{D}}) \in \mathcal{T}^{\texttt{RGB-D}}} \| g(f_{\mathcal{C}}(\mathbf{x}_i^{\mathcal{R}}; \theta_C); \theta_g) - f_{\mathcal{D}}(\mathbf{x}_i^{\mathcal{D}}; \theta_D) \|^2 . \tag{8.3}$$

Finally, $\mathcal{L}_{MMD}$ is the *MMD* [80] loss to minimize the domain shift between the distribution of features extracted from real and synthetic depth images of these training sets:

$$\mathcal{L}_{MMD}(\theta_D; \mathcal{T}^{\mathcal{S}}, \mathcal{T}^{\texttt{RGB-D}}) = \left\| \begin{array}{l} \frac{1}{|\mathcal{T}^{\texttt{RGB-D}}|} \sum_{\mathbf{x}_i^{\mathcal{D}} \in \mathcal{T}^{\texttt{RGB-D}}} \phi(f_{\mathcal{D}}(\mathbf{x}_i^{\mathcal{D}}; \theta_D)) - \\ \frac{1}{|\mathcal{T}^{\mathcal{S}}|} \sum_{\mathbf{x}_i^{\mathcal{S}} \in \mathcal{T}^{\mathcal{S}}} \phi(f_{\mathcal{D}}(\mathbf{x}_i^{\mathcal{S}}; \theta_D)) \end{array} \right\|^2 , \tag{8.4}$$

where $\phi(\cdot)$ denotes the mapping to kernel space, but the exact mapping is typically unknown in practice. By applying the kernel trick, this rewrites to:

$$\begin{aligned} \mathcal{L}_{MMD}(\theta_D; \mathcal{T}^{\mathcal{S}}, \mathcal{T}^{\texttt{RGB-D}}) = & \frac{1}{|\mathcal{T}^{\texttt{RGB-D}}|^2} \sum_{i,i'} k(f_{\mathcal{D}}(\mathbf{x}_i^{\mathcal{D}}; \theta_D), f_{\mathcal{D}}(\mathbf{x}_{i'}^{\mathcal{D}}; \theta_D)) \\ & - \frac{2}{|\mathcal{T}^{\texttt{RGB-D}}||\mathcal{T}^{\mathcal{S}}|} \sum_{i,j} k(f_{\mathcal{D}}(\mathbf{x}_i^{\mathcal{D}}; \theta_D), f_{\mathcal{D}}(\mathbf{x}_j^{\mathcal{S}}; \theta_D)) \\ & + \frac{1}{|\mathcal{T}^{\mathcal{S}}|^2} \sum_{j,j'} k(f_{\mathcal{D}}(\mathbf{x}_j^{\mathcal{S}}; \theta_D), f_{\mathcal{D}}(\mathbf{x}_{j'}^{\mathcal{S}}; \theta_D)) , \end{aligned} \tag{8.5}$$

where $k(\cdot, \cdot)$ denotes a kernel function. In this work, we implement $k(\cdot, \cdot)$ as an RBF kernel, such that:

$$k(\mathbf{x}, \mathbf{y}) = \mathrm{e}^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} , \tag{8.6}$$

where we select the bandwidth $\sigma = 1$. Note that our method is not sensitive to the exact value of σ.

At run-time, given a real color image $\mathbf{x}^{\mathcal{R}}$, we extract its features in color space and map them to the depth feature space by the networks $f_{\mathcal{C}}$ and g, respectively, and then use the pose estimator $h_{\mathcal{D}}$ to predict the 3D pose $\hat{\mathbf{y}}$ of the hand:

$$\hat{\mathbf{y}} = h_{\mathcal{D}}(g(f_{\mathcal{C}}(\mathbf{x}^{\mathcal{R}}))) . \tag{8.7}$$

8.3.2 Network Details and Optimization

For the depth feature extraction network $f_{\mathcal{D}}$, we use a network architecture similar to the 50-layer Residual Network [92], and remove the Global Average Pooling [92] as done in [173, 181, 205]. The convolutional layers are followed by two fully connected layers of

1024 neurons each. The pose estimator $h_{\mathcal{D}}$ consists of one single fully connected layer with 63 outputs, which correspond to 3 values for each of the 21 joints. Additionally, we add a 3D pose prior, as introduced in Chapter 4, as a bottleneck layer to the pose estimator network $h_{\mathcal{D}}$, which was shown to efficiently constrain the 3D hand poses and also gives better performance in our case.

For the color feature extractor $f_{\mathcal{C}}$, we use the same architecture as the depth feature extractor, which makes the feature extractor comparable to the one used in [172, 237].

As in Chapter 7, we use a two Residual blocks [92] network g for mapping the features of size 1024 from color space to depth space. Each fully connected layer within the Residual block has 1024 neurons.

In practice, we use $\beta = 0.02$ and $\gamma = 0.01$ for the meta parameters of the objective function in Eqn. (8.1) for all our experiments. We first pretrain $f_{\mathcal{D}}$ and $h_{\mathcal{D}}$ on the synthetic depth dataset $\mathcal{T}^{\mathcal{S}}$. We also pretrain $f_{\mathcal{C}}$ by predicting depth from color images [60]. Pretraining is important in our experiments for improving convergence. We then jointly train all the networks together using the ADAM optimizer [125] with a batch size of 128 and an initial learning rate of 10^{-4}.

8.4 Evaluation

In this section, we evaluate our method on 3D hand pose estimation from color images only.

8.4.1 Training Data

We use the Stereo Hand Pose Benchmark (STB) [316] and the Rendered Hand Dataset (RHD) [325] for training and testing our approach for 3D hand pose estimation. We follow the protocol of [172, 237, 325], which use the first two sequences from the STB dataset for testing. The remaining ten sequences from the STB dataset together with the RHD dataset are used for the training set $\mathcal{T}^{\texttt{RGB-D}}$, since they both contain aligned depth and color images. Creating synthetic depth images for hands is a relatively simple problem. For generating the training set $\mathcal{T}^{\mathcal{S}}$ we use the publicly available 3D hand model of [271] to render synthetic depth images of a hand. We use 5M synthetic images of the hand that are rendered online during training from poses of the NYU 3D hand pose dataset [271] perturbed with randomly added articulations. Furthermore, [172, 325] align their 3D prediction to the ground truth wrist which we also do for comparison.

We use the pipeline described in Chapter 4 to preprocess the depth images: It crops a 128×128 patch around the hand location, and normalizes its depth values to the range of $[-1, +1]$. For the color image we also crop a 128×128 patch around the corresponding hand location and subtract the mean RGB values. When a hand segmentation mask is available, such as for the RHD dataset [325], we superimpose the segmented hand on

random color backgrounds from the RGB-D dataset [136]. During training, we randomly augment the hand scale, in-plane rotation, and 3D translation, as done in Section 4.2.5.

8.4.2 Comparison with Baselines

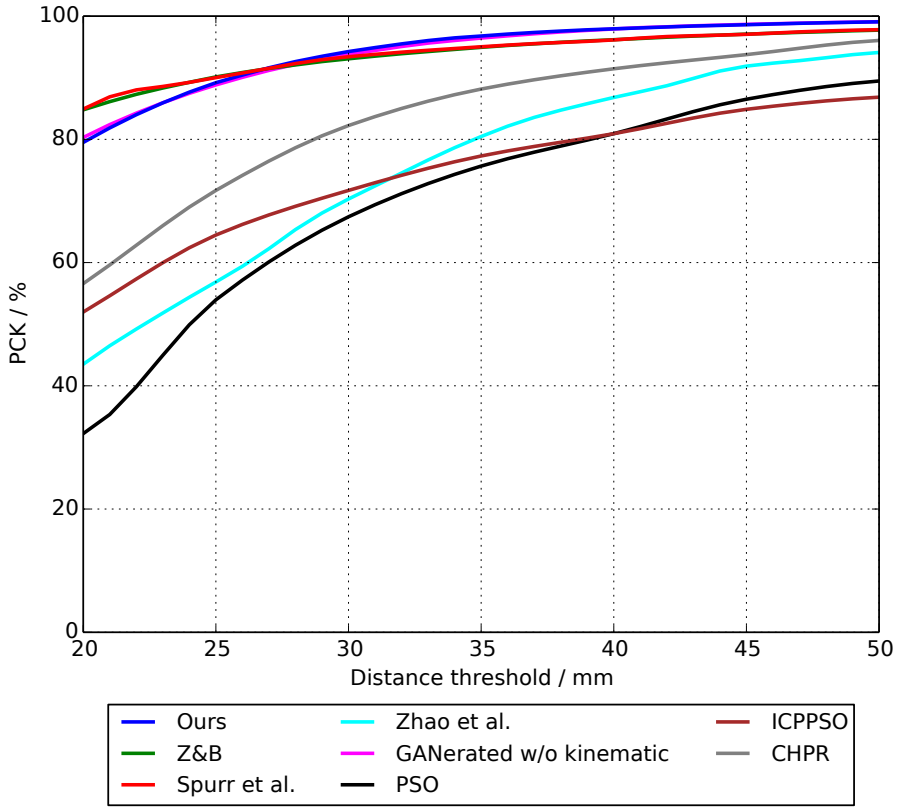


Figure 8.4: 3D Percentage of Correct Keypoints (PCK) curves for comparison to state-of-the-art 3D hand pose estimation methods on the STB dataset [316]. Note that all other approaches use annotated color images for training, whereas we do not use any annotations for the color images.

We compare to the following methods: GANerated [172][2], which uses a *GAN* to adapt synthetic color images for training a Convolutional Neural Network (CNN); Z&B [325], which uses a learned prior to lift 2D keypoints to 3D locations and the similar approach of Zhao *et al.* [319]; Zhang *et al.* [316], which uses stereo images to estimate depth and applies a depth-based pose estimator with variants denoted PSO, ICPPSO, and CHPR; Spurr *et al.* [237], which projects color images to a common embedding that is shared between images and 3D poses.

[2]The results reported in the paper [172] are tracking-based and include an additional inverse kinematics step. In order to make their results more comparable to ours, we denote results predicted for each frame separately without inverse kinematics kindly provided by the authors.

Figure 8.4 shows the *PCK* curves over different error thresholds, which is the most common metric on the STB dataset [172, 237, 316, 325]. This metric denotes the average percentage of predicted joints below an Euclidean distance from the ground truth 3D joint location. While all methods that we compare to require annotations for color images, we can achieve comparable results without annotations of color images.

8.4.3 3D Pose from Predicted Depth Images

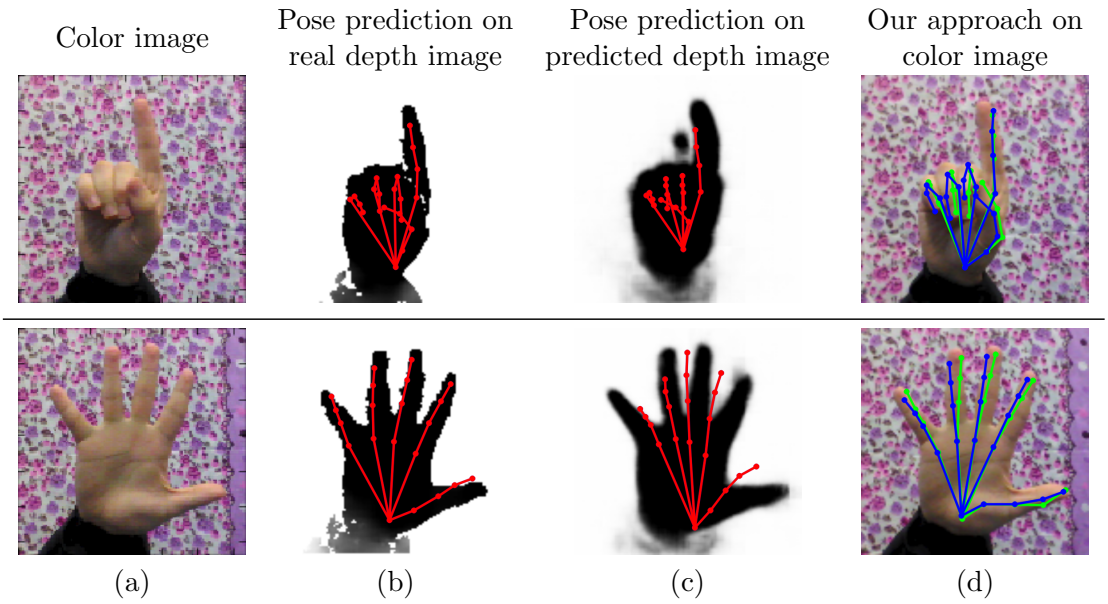


Figure 8.5: We use the paired color and depth images shown in (a) and (b) to predict depth from color images [60] shown in (c). We further apply our 3D pose estimator DeepPrior++, introduced in Section 4.2.5, on these images. These predictions from the predicted and real depth images are shown in (b) and (c), respectively. Although the predicted depth images look visually close, the accuracy of the estimated 3D pose is significantly worse compared to using the real depth images. The results from our method are shown in (d) and provide a significantly higher accuracy. Our predictions are shown in blue and the ground truth in green.

3D hand pose estimation methods work very well on depth images [173, 181]. Since we have paired color and depth images, we can train a *CNN* to predict the depth image from the corresponding color image [60]. Since the pose estimator works on cropped image patches, we only use these cropped image patches for depth prediction, which makes the task easier. We then use the predicted depth image for our depth-based 3D hand pose estimator DeepPrior++. Although this approach also does not require any annotations of color images, our experiments show that this performs significantly worse on the STB dataset compared to our approach. The 3D pose estimator gives an average Euclidean joint error of 17.6 mm on the real depth images and 39.8 mm on the predicted depth images. We show a qualitative comparison in Figure 8.5.

8.4.4 Comparison with Domain Adaptation

Finally, we evaluate the domain adaption technique of [69] that aims to learn invariant features with respect to the shift between the color and depth domains. However, this approach gives an average 3D error of 28.5 mm, which shows that although this technique helps for general applications such as classification, the features are not well suited for 3D pose estimation. For comparison, training on synthetic images only gives an average 3D error of 31.4 mm.

8.4.5 Qualitative Results

We show some qualitative results of our method for 3D hand pose estimation in Figure 8.6. These examples show that our approach predicts very accurate poses.

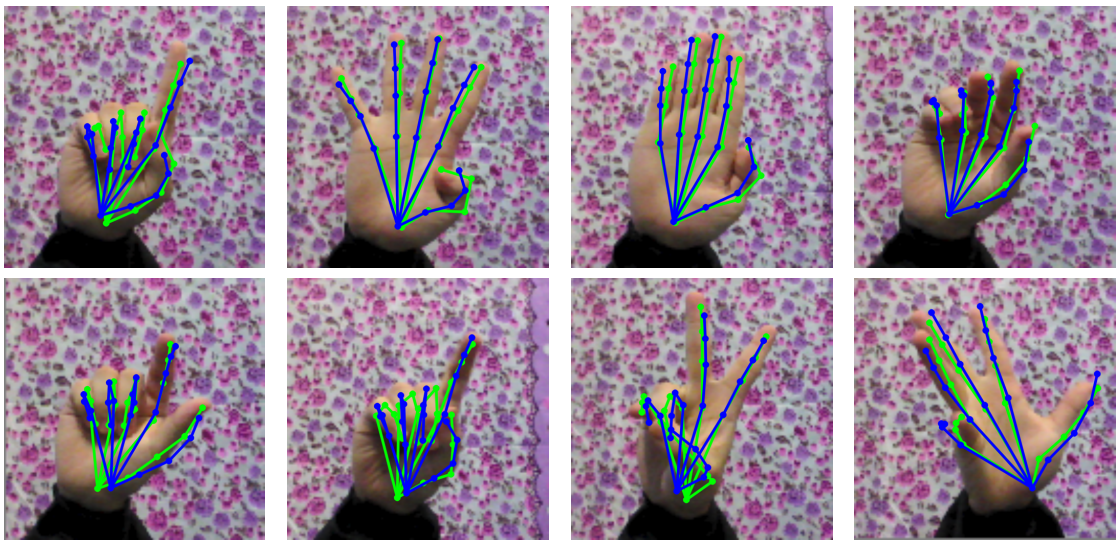
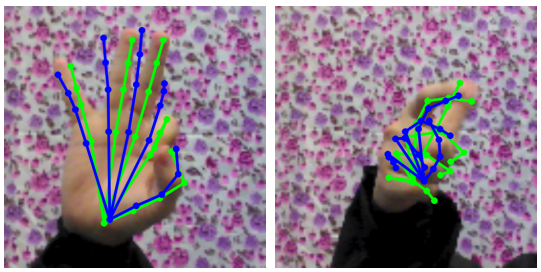


Figure 8.6: Qualitative results of our method for 3D hand pose estimation on the STB dataset [316]. Green denotes ground truth and blue corresponds to the predicted pose.

Figure 8.7 illustrates some failure cases that occur due to the challenges of the test sets, such as misalignment and confusion of the different fingers, severe self-occlusions, or missing poses in the paired dataset $\mathcal{T}^{\text{RGB-D}}$ that can be simply resolved by capturing additional data with an RGB-D camera.

8.4.6 Runtime

All experiments are implemented using Theano [15] and run on an Intel Core i7 with an nVidia Titan. Given an image window extracted around the hand, our approach takes 8.6 ms for 3D hand pose estimation to extract color features, map them to the depth feature space, and estimate the 3D pose.



(a) (b)

Figure 8.7: Some failure cases for 3D hand pose estimation due to (a) misalignment/confusion of the fingers, and (b) severe self-occlusions.

8.5 Discussion

In this chapter we presented a novel approach for 3D pose estimation from color images, without requiring labeled color images for training. We showed that a pose estimator can be trained on a large number of synthetic depth images, and at run-time, given a color image, we can map its features from color space to depth space. We showed that this mapping between the two domains can easily be learned by having corresponding color and depth images captured by a commodity RGB-D camera. Our approach is simple, general, and can be applied to different application domains, such as 3D hand pose estimation. While for this task our approach achieves performances comparable to state-of-the-art methods, it does not require any labels for the color images.

Part III

Hands and Objects

9 Joint Hand-Object Pose Estimation

Contents

In the previous chapters, we introduced methods for accurate 3D hand pose estimation from images and how to deal with the tedious task of acquiring training data. However, in Virtual Reality (VR) and especially in Augmented Reality (AR) applications, the hand is commonly interacting with or manipulating objects. This introduces additional occlusions of the hand and the object that need to be considered. Therefore, we show in this chapter how to generalize our feedback loop approach introduced in Chapter 5 to multiple objects. This generalization gives accurate results for the challenging task of jointly estimating the 3D poses of hands and objects, while hands interact with objects. This is inherently challenging, since the objects introduce additional occlusions and enlarge the joint configuration space. In such a case, we first estimate initial poses for the hand and the object separately, and then fuse these initial predictions within our feedback framework to increase the accuracy of the two poses. For this complex problem, our novel approach works on each frame independently and does not require a good initialization as current tracking-based approaches do [240, 276], while still outperforming the state-of-the-art when using depth images only.

9.1 Introduction

Joint hand-object pose estimation is still an unsolved problem, yet there are plentiful applications in *AR* and *VR*, *e.g.*, in-hand object manipulation, or video game control.

The task at hand is very challenging: It involves estimating the 3D pose of the hand and the 3D pose of the object. Thus in addition to the large number of Degrees of Freedom (DoF) of the hand there are six DoF of the object. Also, there are severe occlusions of the object and the hand from each other in addition to the self-occlusions of the hand.

So far, the methods that are used to solve this problem rely on model-based tracking approaches [187, 240, 276]. They start from an initial pose of the hand and the object and optimize the pose of the hand and the object for a new frame by starting from the pose of the last frame. Although recent approaches can perform this optimization in real-time [240, 276], these approaches are inherently prone to drifting over time that can cause large problems in practice, especially for long sequences or fast movement. In contrast to these joint hand-object pose estimation methods, our approach only requires a single depth camera, works without initialization, without temporal tracking, and does not drift over time.

In this chapter we present, to the best of our knowledge, the first method for single-frame joint hand-object pose estimation. Our approach does not require temporal tracking and works for single depth images. We build upon recent work that we introduced in Chapter 5, which uses a feedback loop for hand pose estimation. In this chapter we generalize this approach to multiple objects, since this method only accounts for a single hand and does not work well in the case of occlusion. Therefore, we introduce an additional feedback loop for the object. Both feedback loops, one for the hand and one for the object, jointly take a synthesized image of the hand and the object from the previous iteration as input and thus learn to be invariant to the occlusions of the other object.

We evaluate our approach on a public dataset for joint hand-object pose estimation and show that our approach can outperform the state-of-the-art when using depth images only. It is also efficient as our implementation runs in real-time on a single GPU.

In the remainder of the chapter, we describe our approach in Section 9.2 and evaluate it in Section 9.3.

9.2 Generalized Feedback Loop for Joint Hand-Object Pose Estimation

We now aim at estimating both the pose $\mathbf{P}^H$ of a hand and the pose $\mathbf{P}^O$ of an object simultaneously while the hand is manipulating the object. We start the pose prediction by first estimating the locations of the hand and the object within the input image. Using the localization, we predict an initial estimate of the pose for the hand and the object separately. In practice, these poses are not very accurate, since hand and object can severely occlude each other. Therefore, we introduce feedback by first synthesizing depth images of the hand and the object and merge them together. An overview of our method with the different operations and networks is shown in Figure 9.1. We then predict an update that aims at correcting the object pose and the hand pose. Each step is

performed by a Convolutional Neural Network (CNN). As in Chapter 5, we denote them localizer *CNN* loc(.), predictor *CNN* pred(.), synthesizer *CNN* synth(.), and updater *CNN* updater(.,.). However, this time we use one network for the hand and one for the object and use suffixes ^{H} and ^{O}, respectively, to distinguish them. In the following we detail how the individual networks are trained.

We now aim at estimating the hand pose and the object pose simultaneously, thus we require for the training set the hand pose and the object pose for each depth frame $\mathcal{T} = \{(\mathcal{D}_i, \mathbf{P}_i^O, \mathbf{P}_i^H)\}_{i=1}^N$. Acquiring these annotations for real data can be very cumbersome, and therefore we only use synthetically generated training data. We outline the generation of the training data in Section 9.2.5.

9.2.1 Learning the Hand and Object Localizers

During experiments we have found that accurate localization of the hand and the object is crucial for accurate pose estimation. Since we use multiple objects, we use the center of mass to perform a rough localization and extract a larger cube around the center of mass that contains the hand and the object. Then, we use two *CNNs* to perform the localization of the hand and the object separately. We train two networks, $\text{loc}^H(.)$ for the hand and $\text{loc}^O(.)$ for the object, both taking the same input image cropped from the center of mass location. The network architectures are the same as the one used in Section 5.3.2 and shown in Figure 4.4. The networks are trained to predict the 2D location and depth of the MCP joint of the hand $\mathbf{l}^H$, and the object centroid $\mathbf{l}^O$, respectively. Formally, we optimize the cost function

$$\widehat{\Gamma} = \arg\min_{\Gamma} \sum_{(\mathcal{D},\mathbf{l}^H)\in\mathcal{T}} \|\text{loc}_\Gamma^H(\mathcal{D}) - \mathbf{l}^H\|^2 \tag{9.1}$$

for the hand localizer, and

$$\widehat{\chi} = \arg\min_{\chi} \sum_{(\mathcal{D},\mathbf{l}^O)\in\mathcal{T}} \|\text{loc}_\chi^O(\mathcal{D}) - \mathbf{l}^O\|^2 \tag{9.2}$$

for the object localizer, where Γ and χ are the parameters of the localizer *CNNs*.

9.2.2 Spatial Transformer and Inverse Spatial Transformer Networks

Once we are given the location of the hand and the object, we use them to crop a region of interest around the hand and the object, respectively, using a *STN*. Note that this makes the crop differentiable and thus trainable end-to-end. The *STN* proposed in [111] was parametrized to work with 2D images. In this work, we apply the *STN* on 2.5D depth images. We estimate the spatial transformation A_θ from the 3D location, calculated from the predicted 2D location and depth. Using the intrinsic camera calibration we denote the projection from 3D to 2D as proj($\cdot$), the 3D bounding box as c, and the 3D location as

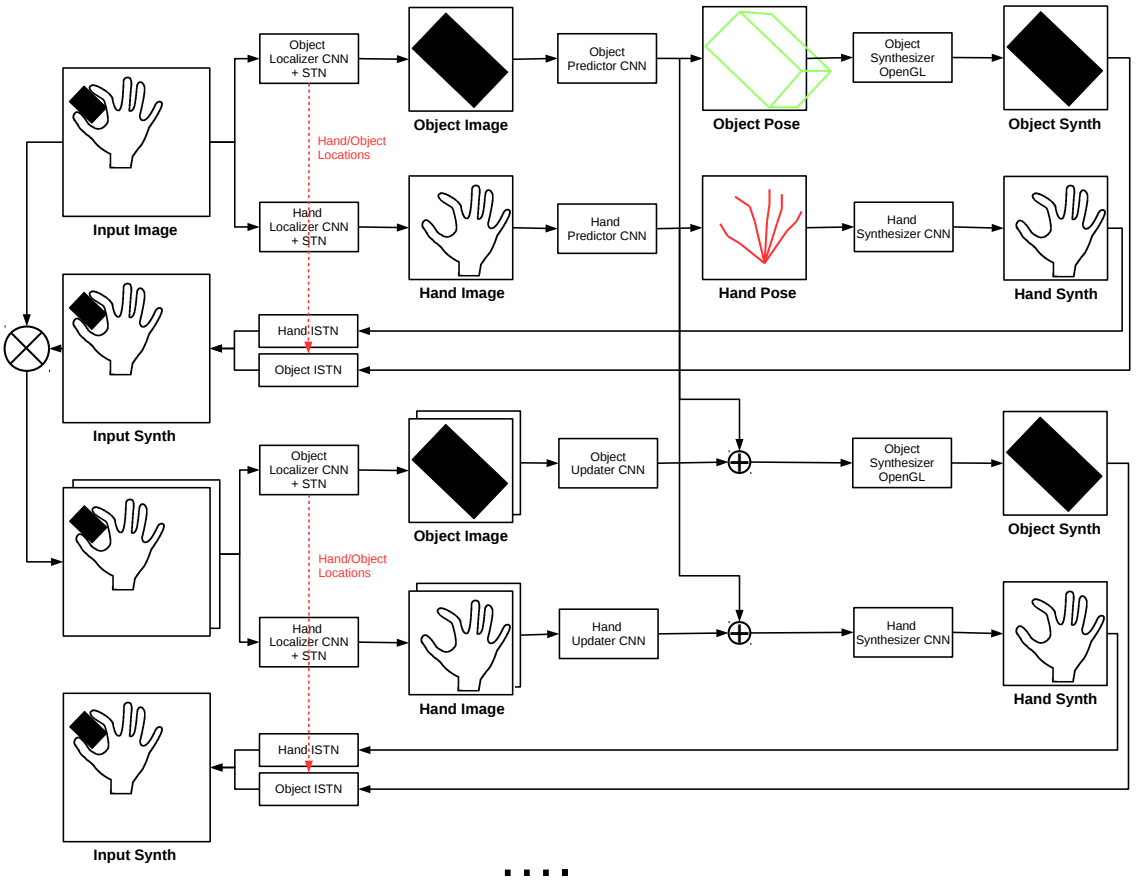


Figure 9.1: Overview of our joint hand-object pose estimation method. We show the predictors for the initial poses with one iteration of the updaters. The input to our method is a crop from the depth camera frame that contains the hand with the object estimated from the center of mass in the depth camera frame. From this input, we localize the hand and the object separately, by predicting the 2D locations and depth using the localizer *CNNs*. We apply Spatial Transformer Networks (STNs) to crop regions of interest around the predicted locations. On these centered crops, we predict initial poses using the predictor *CNNs*, which are then used to synthesize depth images of the hand and the object using the synthesizer *CNNs*. Using Inverse Spatial Transformer Networks (ISTNs), we paste back the different inputs, *i.e.*, the synthesized object and the synthesized hand, into the original input frame. These images are stacked together ($\otimes$) and serve as input to the updater *CNNs* that predict one update for the pose of the hand and one for the object. The updates are added to the poses and the procedure is iterated several times.

$\mathbf{t}$. We can estimate the 2D bounding box projection as $(x^{\pm}, y^{\pm}) = \text{proj}(\mathbf{t} \pm c)$ and define $x^{\Delta} = (x^{+} - x^{-})$ and $y^{\Delta} = (y^{+} - y^{-})$. This leads to the parametrization of the spatial transformation A_{θ} as:

$$A_{\theta} = \begin{pmatrix} x^{\Delta}/2 & 0 & x^{-} + x^{\Delta}/2 \\ 0 & y^{\Delta}/2 & y^{-} + y^{\Delta}/2 \\ 0 & 0 & 1 \end{pmatrix} \quad , \tag{9.3}$$

for the parametrization of the sampling grid locations as in [111]. The source sampling grid locations (x^s, y^s) are obtained by transforming the regularly sampled target grid locations (x^t, y^t) as:

$$(x^s, y^s)^T = A_{\theta} \cdot (x^t, y^t, 1)^T \quad . \tag{9.4}$$

We then use bilinear sampling to interpolate the depth values of input depth image $\mathcal{D}$ of size $W \times H$:

$$\text{STN}(\mathbf{l}, \mathcal{D}) = \sum_{h}^{H} \sum_{w}^{W} \mathcal{D}[h, w] \max(0, 1 - |x^s - w|) \max(0, 1 - |y^s - h|) \quad . \tag{9.5}$$

Since we apply a detection and crop using the *STN*, we require the inverse operation for putting the synthesized object back into the original image frame before applying the updater *CNN*. Hence, we apply an *ISTN* [148]. It operates the same way as the *STN*, but uses the inverse of the spatial transformation A_{θ}.

9.2.3 Learning the Hand and Object Pose Predictors

Once we are given an accurate location of the hand and the object, we crop a patch around the hand and the object using the *STN* and use the crop to estimate the pose. We denote c^H the crop of the hand and c^O the crop of the object. We denote this operation as $c^O = \text{STN}(\text{loc}^O(\mathcal{D}), \mathcal{D})$. For the pose prediction, we require appropriate parametrization of the hand and object poses. For the object pose $\mathbf{P}^O$, we use the eight corners of its 3D bounding box, which was shown to perform better than rotation and translation [297]. For the hand pose, we use the 3D joint locations with a prior on the 3D hand poses, as introduced in Chapter 4. A prior on the 3D hand pose is very effective in case of occlusions, since it constraints the predicted hand pose to valid poses. For our experiments with hands manipulating objects, we use a similar architecture to the network shown in Figure 4.4 for the sake of speed. However, for the hand predictor *CNN* $\text{pred}^H(.)$ we use $3 \cdot J$ neurons for the output layer and for the object predictor *CNN* $\text{pred}^O(.)$ we use $3 \cdot 8$ neurons.
Again, we optimize two cost functions to train the predictor *CNNs*. For the hand pose predictor we optimize

$$\widehat{\eta} = \arg\min_{\eta} \sum_{(\mathcal{D}, \mathbf{P}^H) \in \mathcal{T}} \|\mathcal{P}(\text{pred}^H_{\eta}(c^H)) - \mathbf{P}^H\|^2 \quad , \tag{9.6}$$

where $\mathcal{P}$ is the linear prior on the 3D hand poses. For the object pose predictor we optimize

$$\widehat{\Pi} = \arg\min_{\Pi} \sum_{(\mathcal{D},\mathbf{P}^O)\in\mathcal{T}} \|\text{pred}^O_\Pi(c^O) - \pi(\mathbf{P}^O)\|^2 \ , \tag{9.7}$$

where $\pi(\cdot)$ returns the 3D bounding box coordinates given the object pose. η and Π denote the parameters of the predictor *CNNs*. We use orthogonal Procrustes analysis [117] to obtain the 3D object pose from the output of $\text{pred}^O(.)$.

9.2.4 Learning the Hand and Object Updaters

Since we now have initial estimates of the pose for the hand and the object, we can use them to start a feedback loop that considers the interaction of hand and object. Therefore, we extend the input data of the updater *CNN* with the synthesized image of the hand and the object given the initially predicted poses. This requires training a synthesizer *CNN* or using a software renderer. Since we have the 3D model of the object readily available, we implement the object synthesizer $\text{synth}^O(.)$ with an OpenGL-based renderer. During our experiments, we also evaluated training the updater *CNN* with the object synthesizer *CNN* and this resulted in similar results compared to the OpenGL-based rendering. The hand synthesizer *CNN* $\text{synth}^H(.)$ is trained in the same way as described in Section 5.3.4. Also, the network architecture of the updater *CNN* is the same as shown in Figure 5.12.

For training the updater *CNN*, the synthesized images are merged into a single depth image:

$$S = \min(\text{ISTN}(\text{loc}^H(\mathcal{D}), \text{synth}(\mathbf{P}^H)), \text{ISTN}(\text{loc}^O(\mathcal{D}), \text{synth}(\mathbf{P}^O))) \otimes \mathcal{D} \ , \tag{9.8}$$

where $\min(A, B)$ denotes the pixel-wise minimum between two depth images A and B, and $\otimes$ denotes a channel-wise stacking. We then train the updater *CNN* for the hand $\text{updater}^H(.,.)$ and the object $\text{updater}^O(.,.)$ by minimizing the following cost function:

$$\sum_{(\mathcal{D},\mathbf{P}^H,\mathbf{P}^O)\in\mathcal{T}} \sum_{\mathbf{P}'^O\in\mathcal{T}_O} \sum_{\mathbf{P}'^H\in\mathcal{T}_H} \max(0, \|\mathbf{P}'' - \mathbf{P}\| - \lambda\|\mathbf{P}'^{H,O} - \mathbf{P}\|) \ , \tag{9.9}$$

where $\mathbf{P}'' = \mathbf{P}'^{H,O} + \text{updater}^{H,O}(c^{H,O}, S)$, and $\mathcal{T}_H$ and $\mathcal{T}_O$ are sets of poses for the hand and the object, respectively. When optimizing Eqn. (9.9) for the hand updater *CNN* $\text{updater}^H(.,.)$, the variables with suffix ^{H} are used, and when optimizing for the object updater *CNN* $\text{updater}^O(.,.)$ the variables with suffix ^{O} are used.
When training the updater *CNN* $\text{updater}^H(.,.)$ for the hand, $\mathcal{T}_H$ is initialized and updated during training as described in Section 5.3.5. $\mathcal{T}_O$ is made from random poses around the ground truth object pose and poses from the predictor $\text{pred}^O(.)$ applied to training images.
When training the updater *CNN* $\text{updater}^O(.,.)$ for the object, $\mathcal{T}_O$ is defined as in Section 5.3.5. $\mathcal{T}_H$ is made from random poses around the ground truth together with predic-

tions from the training data.

For inference, we iterate the updater *CNN* several times: We first obtain the initial estimates $\mathbf{P}^{(0)}$ for hand H and object O, by running the predictor *CNN* on the cropped locations from the localizer *CNN*. Then we predict the updates using the merged image $S^{(i)}$ from the previous iteration. Figure 9.2 gives a formal expression of this algorithm.

$$
\begin{cases}
\widehat{\mathbf{P}}^{(0),H} \leftarrow \text{pred}^{H}(\text{STN}(\text{loc}^{H}(\mathcal{D}), \mathcal{D}) \\
\widehat{\mathbf{P}}^{(0),O} \leftarrow \text{pred}^{O}(\text{STN}(\text{loc}^{O}(\mathcal{D}), \mathcal{D}) \\
S^{(i)} \leftarrow \min(\text{ISTN}(\text{loc}^{H}(\mathcal{D}), \text{synth}(\widehat{\mathbf{P}}^{(i),H})), \text{ISTN}(\text{loc}^{O}(\mathcal{D}), \text{synth}(\widehat{\mathbf{P}}^{(i),O}))) \otimes \mathcal{D} \\
\widehat{\mathbf{P}}^{(i+1),H} \leftarrow \widehat{\mathbf{P}}^{(i),H} + \text{updater}^{H}(\text{STN}(\mathbf{P}^{(i),H}, \mathcal{D}), S^{(i)}) \\
\widehat{\mathbf{P}}^{(i+1),O} \leftarrow \widehat{\mathbf{P}}^{(i),O} + \text{updater}^{O}(\text{STN}(\mathbf{P}^{(i),O}, \mathcal{D}), S^{(i)}) \quad .
\end{cases}
$$

Figure 9.2: Algorithm for joint hand-object pose estimation.

9.2.5 Training Data Generation

Capturing real training data of a hand manipulating an object can be very cumbersome, since in our case it requires 3D hand pose and 3D object pose annotations for each frame. This is hindered by severe occlusions, or not always possible at all. Our approach leverages annotations from datasets of hands in isolation, which are much simpler to capture in practice [201, 247, 254], and data from 3D object models, which are available at large scales [35]. We use an OpenGL-based rendering of the 3D object model and fuse the rendering with the frames from a 3D hand dataset. Since we use depth images, we can simply take the minimum of both depth images for each pixel. We render the object on top of the hand, placing the object near the fingers, and apply a simple collision detection, such that the object is not placed "within" the hand point cloud. The object poses, *i.e.*, rotations and translations, are sampled randomly, constrained by the collision detection. In order to account for sensor noise, we add small Gaussian noise to the rendering. In Figure 9.3 we show some samples of synthesized training data.

Besides this semi-synthetic training data, we also use completely synthetic training data of hands and objects. We therefore use a marker-based motion capture system [88] to capture the hand and object pose while the hand is manipulating different objects. Since the image data is corrupted with the markers, we render a synthetic hand model and the CAD object model given the captured poses, and use this rendering as additional training data.

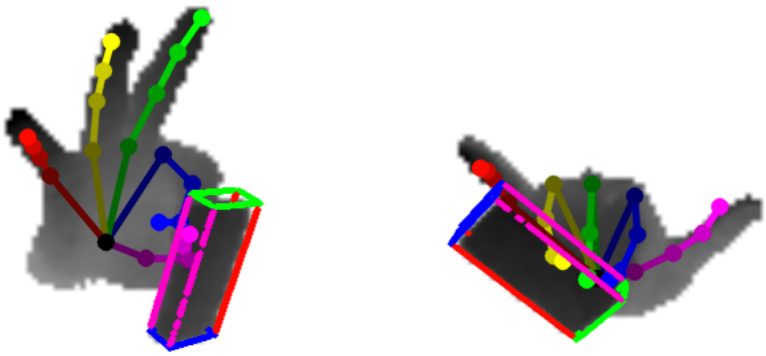


Figure 9.3: Training samples used for joint hand-object pose estimation. The interaction between hand and object cannot be *accurately* modeled easily, but the solution space of possible object locations and possible object poses can be significantly reduced, which enables training our proposed method.

9.3 Evaluation

In this section we present the evaluation of our approach for joint hand-object pose estimation on the challenging DexterHO dataset [240]. First, we describe the data we use for training. Then, we evaluate our method qualitatively and quantitatively.

9.3.1 Training Data

Since there are no datasets for joint hand-object pose estimation available that contain enough samples to train a Neural Network, our approach relies on fusing real and synthetic data. We use real hand data and synthetic object data as described in Section 9.2.5. We use real hand data from the large MSRA [247] dataset, consisting of 72k depth frames of hands of 9 different subjects and a wide variety of hand poses. Further, we use the dataset of Qian *et al.* [201], which contains 2k depth frames of 6 different subjects. Both datasets were captured using a Creative RealSenz Time-of-Flight camera, which is the same camera as used for the experiments on the benchmark dataset. The depth image resolution is 320×240 and the annotations contain $J = 21$ joint locations. The 3D object models are manually created from simple geometric primitives to resemble the objects from the benchmark dataset.

9.3.2 Comparison with Baselines

We evaluate the approach using the combined hand-object metric proposed by [240] and discussed in Section 3. Although we predict the 3D location of all joints of the hand,

Method	Sequence	Rigid	Rotate	Occlusion	Grasp1	Grasp2	Pinch	Average
Sridhar *et al.* [240] RGB+D	Finger tips	14.2 mm	16.3 mm	17.5 mm	18.1 mm	17.5 mm	10.3 mm	15.6 mm
	Object corners	13.5 mm	26.8 mm	11.9 mm	15.3 mm	15.7 mm	13.9 mm	16.2 mm
	Combined	14.1 mm	**18.0 mm**	16.4 mm	**17.6 mm**	**17.2 mm**	**10.9 mm**	**15.7 mm**
Sridhar *et al.* [240] D only	Combined	–	–	–	–	–	–	18.5 mm
This work init D only	Combined	14.1 mm	20.3 mm	18.8 mm	20.8 mm	24.3 mm	17.6 mm	19.3 mm
This work feedback D only	Finger tips	14.2 mm	17.9 mm	16.3 mm	22.7 mm	24.0 mm	18.5 mm	18.9 mm
	Object corners	8.4 mm	23.4 mm	7.4 mm	8.2 mm	16.6 mm	9.6 mm	12.4 mm
	Combined	**13.2 mm**	18.9 mm	**14.5** mm	20.2 mm	22.5 mm	16.7 mm	17.6 mm

Table 9.1: Quantitative results on the DexterHO dataset [240]. Note, that [240] uses color and depth information together with a tracking-based approach, which relies on a strong pose prior from the previous frame. In comparison, we perform hand and object pose estimation for each frame independently. Also, we use depth information only, for which their reported average error is larger than for our approach. Our approach significantly outperforms the baseline for the accuracy of the object pose on average.

we only use the 3D locations of the finger tips for calculating the error metric. Also, we predict the full 6 *DoF* pose of the object and calculate the 3D corner locations for the evaluation.

We compare our results to the state-of-the-art on the DexterHO dataset [240] in Table 9.1. Note that the baseline of [240] uses a tracking-based approach, which does not rely on training data but requires the pose of the previous frame as initialization. By contrast, we do not require any initialization at all and predict the poses from scratch on each frame independently. We outperform this strong baseline on two sequences (*Rigid* and *Occlusion*), and when [240] use only depth, as we do, we outperform their method on average over all sequences of the dataset. Our method is significantly more accurate for the object corner metric, since it is much easier to acquire training data for objects compared to hands. For the accuracy of the hand pose estimation, our approach is mostly restricted by the limited training data.

The updaters perform two iterations for the hand and the object, since the results do not improve much for more iterations. Using the feedback loop improves the accuracy on all sequences by 10% on average compared to our initialization.

9.3.3 Qualitative Results

Figure 9.4 shows some qualitative examples. Our approach estimates accurate 3D poses for the hand and the object.
Figure 9.5 shows the pose for consecutive iterations. The predictor provides an initial estimate of the pose of the hand and the object, and our feedback loop improves these initial poses iteratively.

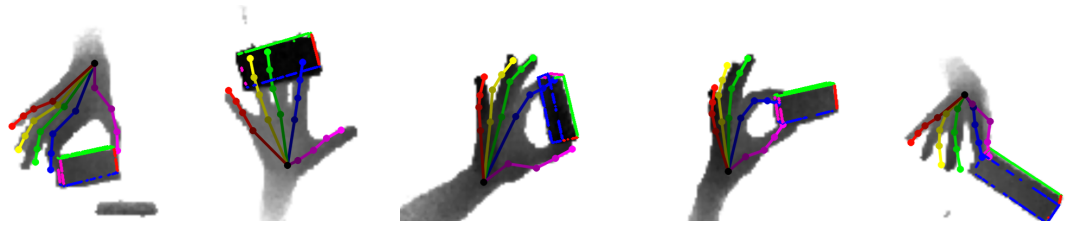


Figure 9.4: Qualitative results on the DexterHO dataset [240]. Our method provides accurate hand and object poses, also when the hand is manipulating an object.

9.3.4 Runtime

Our method is implemented in Python using the Theano library [15] and we run the experiments on a computer equipped with an Intel Core i7, 64GB of RAM, and an nVidia GeForce GTX 980 Ti GPU. Training takes about ten hours for each *CNN*. The runtime is composed of the localizer *CNN* that takes 0.8 ms, the predictor *CNN* with the simpler network architecture takes 0.8 ms, the updater *CNN* takes 1.2 ms for each iteration, and that already includes the synthesizer *CNN* with 0.8 ms. For the joint hand-object pose estimation we have to run each network once for the hand and once for the object. We run the inference in two threads in parallel, one for the hand and one for the object. In practice we iterate our updater *CNN* twice, thus our method runs at over 40 frames per second (fps) on a single GPU.

9.4 Discussion

In this chapter we presented a novel approach for joint hand-object pose estimation. First, we separated the problem into hand pose and object pose estimation, in order to obtain an initial pose for the hand and the object independently. Then, we introduced a feedback loop that refines these initial estimates. Remarkably, our approach does not require real data of hand-object interaction and can be trained on synthetic data, which simplifies the creation of the dataset. We evaluated our approach on a public dataset for joint hand-object pose estimation, where our approach outperformed the state-of-the-art tracking-based approach when using only depth images.

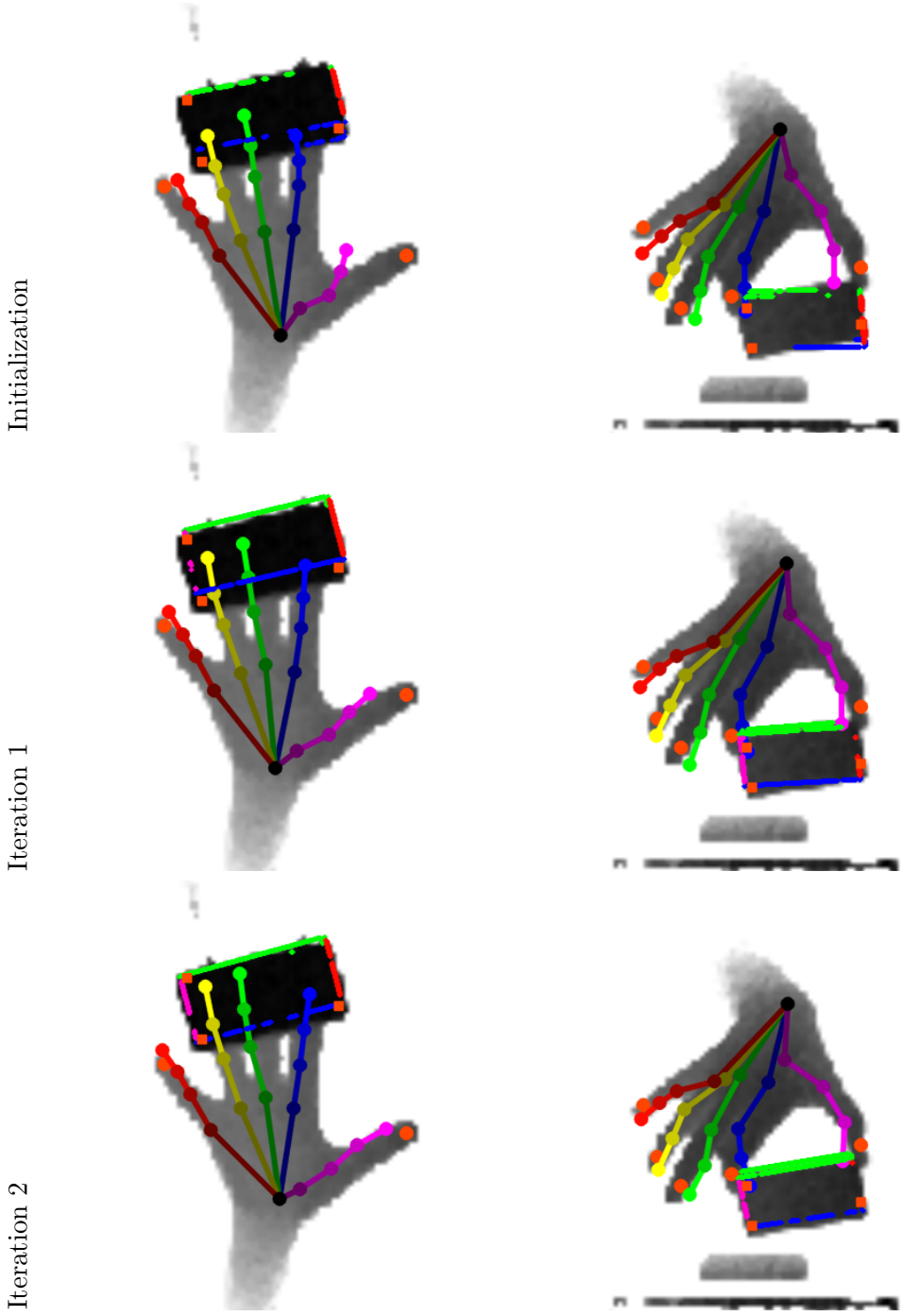


Figure 9.5: Different iterations on the DexterHO dataset [240]. Top row shows the initialization and further rows the consecutive iterations. Our results are shown in color and the ground truth annotations are shown in orange. The ground truth contains the finger tips for the hand and the corners for the object.

10

Occlusion Tolerant 3D Object Pose Estimation

Contents

In the previous chapter, we presented an approach for joint hand-object pose estimation. The assumption for this approach is that we have *a priori* knowledge of the object and the hand available. In practice, however, we do not know the object and possible occluders in advance. Thus, if we do not have any knowledge of the occluder, the accuracy of the predicted pose is worse, as we show in this chapter. Here we focus on the problem of 3D object pose estimation, since the problem is better studied and has more publicly available datasets. Further, it is easier to study this problem for rigid objects compared to the hand, since rigid objects are not articulated. Especially for hand-held objects, the occlusions from the hand have a considerable impact as shown in Figure 10.1. Large regions of the object are occluded and since there are many different ways of grasping or holding an object, it would be difficult to model all different grasps and occlusions during training.

Therefore, we introduce a novel method for robust and accurate 3D object pose estimation from a single color image under large occlusions. Following recent approaches [203, 264], we first predict the 2D projections of 3D points related to the target object and then compute the 3D pose from these correspondences using a geometric method. Unfortunately, as the results of our experiments show, predicting these 2D projections using a regular Convolutional Neural Network (CNN) or a Convolutional Pose Machine (CPM) [290] is highly sensitive to partial occlusions, even when these methods are trained with partially

occluded examples. Our solution is to predict heatmaps from multiple small patches independently and to accumulate the results to obtain accurate and robust predictions. Training subsequently becomes challenging because patches with similar appearances but different positions on the object correspond to different heatmaps. However, we provide a simple yet effective solution to deal with such ambiguities. We show that our approach outperforms existing methods on two challenging datasets: The Occluded LineMOD dataset and the YCB-Video dataset, both exhibiting cluttered scenes with highly occluded objects.

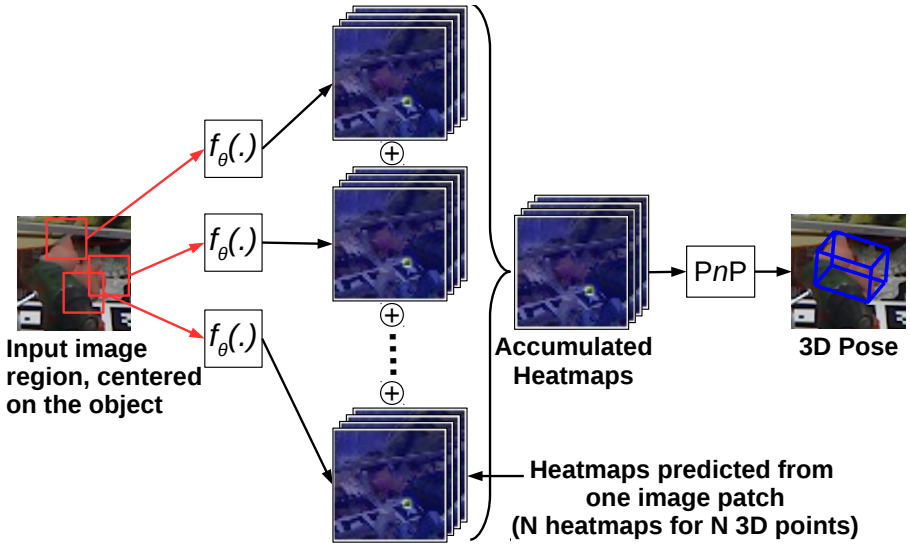
Figure 10.1: Objects from the YCB dataset [32] held in-hand. There are significant occlusions of the object caused by the hand. Since there are many different ways of grasping the objects, it would be difficult to consider all different grasps for creating a training dataset.

10.1 Introduction

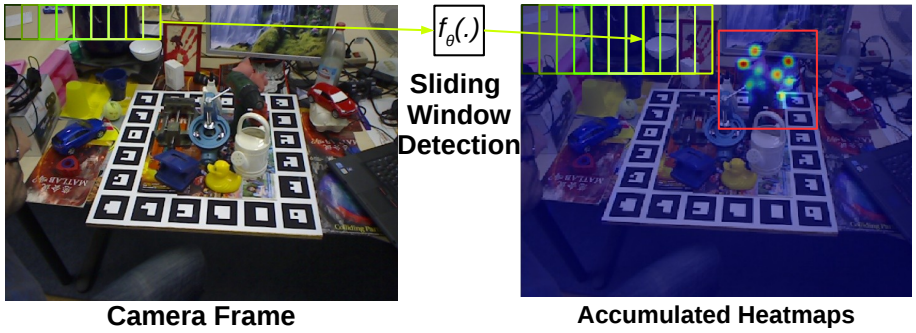
3D object pose estimation from images is an old but currently highly researched topic, mostly due to the advent of Deep Learning-based approaches and the possibility of using large datasets for training such methods. 3D object pose estimation from RGB-D already has provided compelling results [112, 120, 130, 297], and the accuracy of methods that only require RGB images recently led to huge progress in the field [26, 112, 120, 195, 203, 264, 297]. In particular, one way to obtain an accurate pose is to rely on a Deep Network to initially predict the 2D projections of some chosen 3D points and then compute the 3D pose of the object using a PnP method [90]. Such an approach has been shown to be more accurate than the approach of directly predicting the pose used in [195, 203, 264], and, therefore, we used the former approach in the research described in this chapter.

However, while Deep Learning methods allow researchers to predict the pose of fully visible objects, they suffer significantly from occlusions, which are very common in practice: Parts of the target object can be hidden by other objects or by a hand interacting with the object. A common *ad hoc* solution is to train the network with occluded objects in the training data. As the results of our experiments presented in this chaper show, the presence of large occlusions and unknown occluders still decrease the accuracy of the predicted pose.

Instead of using the entire image of the target object as input to the network, we



(a) 3D pose estimation.



(b) Detection.

Figure 10.2: Overview of our method. (a) Given an image region centered on the target object, we sample image patches from which we predict heatmaps for the 2D projections of the corners of the object's 3D bounding box. This prediction is done by a Deep Network $f_\theta(\cdot)$. We aggregate the heatmaps and extract the global maxima for each heatmap, from which we compute the 3D object pose using a PnP algorithm. *We show that $f_\theta(\cdot)$ can be trained simply and efficiently despite the ambiguities that may arise when using small patches as input.* (b) To obtain the image region centered on the object, we apply the predictor in a sliding window fashion and accumulate the heatmaps for the full camera frame. We keep the image region with the largest values after accumulation.

consider image patches, as illustrated in Figure 10.2, since at least some of these are not corrupted by the occluder. Using an image patch as input, our approach learns to predict heatmaps over the 2D projections of 3D points related to the target object. By combining the heatmaps predicted from many patches, we obtain an accurate 3D pose even if some patches actually lie on the occluder or the background instead of on the object. We show results of our method in Figure 10.3.

When moving to an image patch as input, the prediction becomes multimodal. This is

(a) Cat (pink). **(b)** Duck (yellow). **(c)** Can (white). **(d)** Holepuncher (blue). **(e)** Driller (green).

Figure 10.3: Our method can be used to predict the 3D pose of objects even when heavy occlusions are present in color images. We show different objects from the Occluded LineMOD dataset [25]. The green bounding boxes correspond to the ground truth poses and the blue bounding boxes, to our estimated poses.

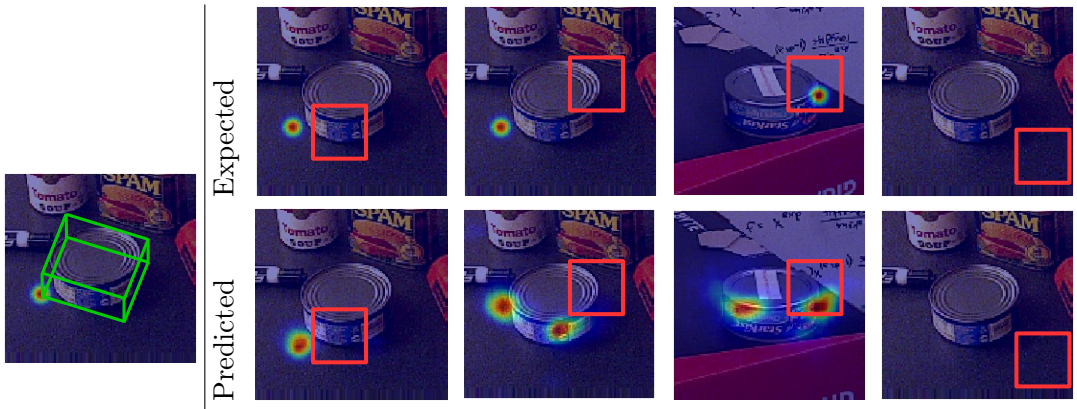


Figure 10.4: Predicting heatmaps from image patches. In this example, we consider predicting the projection of the 3D corner highlighted in (a) for the *tuna fish can* object of the YCB-Video dataset [297]. The red boxes show the input patch of the predicted heatmap. (b) shows a patch from which the projection can be predicted unambiguously. (c) shows two patches that are located in two different positions on the can (notice that the can is flipped and rotated between the two images) while having a similar appearance. In presence of such patches, it is only possible to predict a distribution over the possible locations for the projection. (d) shows a patch on the background, from which we predict a uniform heatmap as it does not provide additional information. See text for details.

shown in Figure 10.4: Some patches may appear on different parts of the target object but look similar. These patches are ambiguous, as they can correspond to different predictions. In such a case, we would like to predict heatmaps with multiple local maxima, one for each possible prediction. The main challenge is that the ambiguities are difficult to identify: This would require us to identify the patches that have similar appearances, from all the possible viewpoints and at all the possible positions on the object.

The authors of [286] faced a similar problem in the context of 2D object detection when aiming to localize semantic parts from feature vectors of a convolutional layer computed by a *CNN*. As we discuss in Section 10.2, the method they proposed is complex both for training and inference, and also inaccurate. The solution we propose is much simpler yet efficient: We train a network to predict heatmaps corresponding to a single solution for training image patches using a least-squares loss function. Thanks to the properties of the least-squares loss, this makes the network naturally predict the *average* of the possible heatmap solutions for a given patch. This is exactly what we want, because it is the best information we can obtain from a single patch even if the information remains ambiguous. We then follow an ensemble approach and take the average of the heatmaps predicted for many patches, which allows us to resolve the ambiguities that arise with individual patches. We finally extract the global maximum from this average as the final 2D location.

Our main contribution is, therefore, a simple method that can be used to accurately predict the 3D pose of an object under partial occlusion. We also considered applying Transfer Learning to exploit additional synthetic training data and improve performances. However, as we show, if the input to a network contains an occluder, the occlusion significantly influences the network output even when the network has been trained with occlusion examples and simply adding more training data does not help. In our case, some of the input patches used by our method will not contain occluders, and Transfer Learning becomes useful. In practice, we use the Feature Mapping method described in Chapter 7, which can be used to map the image features extracted from real images to corresponding image features for synthetic images. This step is not needed for our method to outperform the state-of-the-art but allows us to provide an additional performance boost.

In the remainder of this chapter, we first review related work, then discuss the influence of occlusions on *CNNs*, present our approach, and finally evaluate it and compare it to the state-of-the-art methods on the Occluded LineMOD [25] and the YCB-Video [297] datasets.

10.2 Related Work on Robust 3D Object Pose Estimation

The literature on 3D object pose estimation is extremely large. After the popularity of edge-based [89] and keypoint-based methods [155] waned, Machine Learning and Deep Learning became popular in recent years for addressing this problem [26, 45, 112, 120, 195, 203, 264, 297]. Here, we mostly focus on recent work based on RGB images. In the Evaluation section, we compare our method to recent methods [112, 203, 264, 297].

[26, 112] proposed a cascade of modules, whereby the first module localizes the target objects, and the second module regresses the object surface 3D coordinates. These coordinates then are used to predict the object pose through hypotheses sampling with a pre-emptive RANSAC [90]. Most importantly, we do not directly predict 3D points but average 2D heatmaps. Predicting 3D points for corresponding 2D points seems to be much

more difficult than predicting 2D points for 3D points, as discussed in [297]. Also, surface coordinates are not adapted to deal with symmetric objects. [203] also first detects the target object, then predicts the 2D projections of the corners of the object's 3D bounding boxes and, finally, estimates the 3D object pose from their 3D correspondences using a PnP algorithm. [264] integrated this idea into a recent object detector [208] to predict 2D projections of the corners of the 3D bounding boxes, instead of a 2D bounding box. Similarly, [195] predicts 2D keypoints in the form of a set of heatmaps as we do in this work. However, it uses the entire image as input and, thus, performs poorly on occlusions. It also requires training images annotated with keypoint locations, while we use virtual 3D points. [105] also relies on 2D keypoint detection. They consider partially occluded objects for inferring the 3D object location from these keypoints. However, their inference adopts a complex model fitting and requires the target objects to co-occur in near-regular configuration.

[120] extended the SSD architecture [152] to estimate the objects' 2D locations and 3D rotations. In a next step, they use these predictions together with pre-computed information to estimate the object's 3D pose. However, this requires a refinement step to get an accurate pose, which is influenced by occlusions. [297] segments the objects and estimates their 3D poses by predicting the translation and a quaternion for the rotation, refined by Iterative Closest Point (ICP). Segmenting objects makes their approach robust to occlusions to some extent, however, it requires the use of a highly complex model. [45] considers object parts in order to handle partial occlusions by predicting a set of 2D-3D correspondences from each of these parts. However, the parts have to be manually picked, and it is not clear which parts can represent objects such as those we evaluate in this paper.

As mentioned in the introduction, our method is related to [286]. In the context of 2D object detection, [286] localizes semantic parts from neighboring feature vectors using a spatial offset map. The offset maps are accumulated in a training phase. However, they need to be able to identify which feature vectors support a semantic part from these maps, and complex statistical measures are used to identify such vectors. Our method is significantly simpler, as the mapping between the input patches and the 2D projections does not have to be established explicitly.

[191] already evaluated *CNNs* trained on occlusions in the context of 2D object detection and recognition, and proposed modifying training to penalize large spatial filter support. This yields better performance; however, this does not fully cancel out the influence of occlusions. Some recent work also describes explicitly how to handle occlusions for 3D pose estimation when dealing with 3D or RGB-D data: Like us, [28] relies on a voting scheme to increase robustness to occlusions; [167] first segments and identifies the objects from an RGB-D image. They then perform an extensive randomized search over possible object poses by considering physical simulation of the configuration. [315] combines holistic and local patches for object pose estimation, using a codebook for local patches and applying a nearest-neighbor search to find similar poses, in a way similar to [56, 121]. In

contrast to these methods, we use only color images.

Our method is also related to ensemble methods and, in particular, the Hough Forests [68], which are based on regression trees. Hough Forests also predict 2D locations from multiple patches and are multimodal. Multimodal prediction is easy to perform with trees, as the multiple solutions can be stored in the tree leaves. With our method, we aim to combine the ability of Hough Forests for multimodal predictions and the learning power of Deep Learning. [211] already reformulated a Hough Forest as a *CNN* by predicting classification and regression for patches of the input image. However, this method requires to handle the detection separately, and each patch regresses a single vector, which is not multimodal and requires clustering of the predicted vectors. In this paper, we show that carrying out a multimodal prediction with Deep Networks to address our problem is, in fact, simple.

10.3 Influence of Occlusions on Deep Networks

In this section, we describe how we evaluate how much a partial occlusion influences a Deep Network, whether it is a standard *CNN* or a *CPM* [290]. Specifically, a *CPM* is a carefully designed *CNN* that predicts dense heatmaps by sequentially refining results from previous stages. The input features are concatenated to intermediary heatmaps in order to learn spatial dependencies.

For this experiment, depicted in Figure 10.5, we use an image centered on an object as input to a network—here, the *Cat* object from the Occluded LineMOD dataset [25]. We then compare the layer activations in the absence of occlusion, and when the object is occluded by an artificial object (here, a striped triangle). We consider two networks: A standard *CNN* trained to predict the 2D projections of 3D points as a vector [203], and a *CPM* [290] with three stages trained to predict a heatmap for each of the same 2D projections. For the 3D points, we use the corners of the 3D bounding box of the object.

As can be seen in Figure 10.5, the occlusion induces changes in the activations of all the layers of both networks. For a standard *CNN*, the occlusion spreads to more than 20% in the last feature map, and, beyond the first fully connected layer, more than 45% of all activations are changed. In this case, all the predictions for the 2D projections, occluded or not, are inaccurate. A similar effect can be observed for the *CPM*: Here, the altered activations are more specifically localized to the occluded region due to the convolutions, with more than 29% of the activations changed in the last feature map. In this case, the predictions of the 2D projections are inaccurate when the 3D points are occluded. When the 3D points are not occluded, the predicted projections are sometimes correct, because the influence of the occluder spreads less with a *CPM* than with a standard *CNN*.

Figure 10.6 shows a more detailed visualization of the effect of occlusions on the feature maps of *CNNs*. We show two examples: in the top part of the figure an example with small occlusion and in the bottom part of the figure an example with a larger occlusion. In each part of the figure, we show the feature maps with and without occlusions together

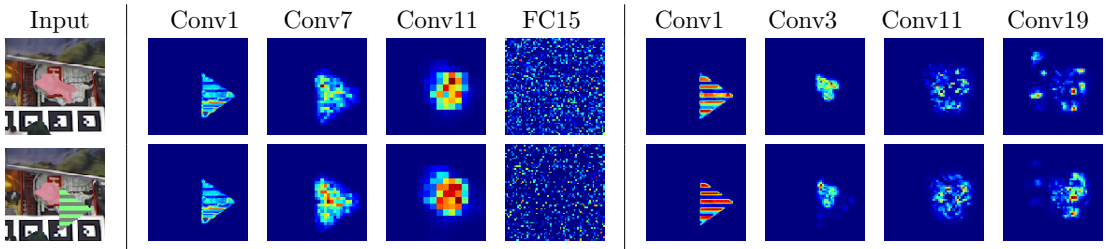


Figure 10.5: Effect of occlusions on the feature maps of Deep Networks. (a) Input image without (top) and with (bottom) partial occlusion. (b-Top) Sums of absolute differences between the feature maps with and without occlusions for a *CNN* trained without occlusions. (b-Bottom) Same when the network is trained with occlusion examples. (c) Same for a *CPM*. The influence of the occlusion increases with the layers' depths, as receptive fields are larger in the deeper layers than in the first layers, even when the method is trained with occlusion examples.

with the sums of the absolute differences between them. Further, we compare when the network is trained with or without occlusion examples. The left column shows the results for a standard *CNN* [203] and the right column shows the same for a *CPM* [290]. The influence of the occlusion increases with the layers' depths, as receptive fields are larger in the deeper layers than in the first layers, even when the method is trained with occlusion examples.

10.4 Minimizing the Effect of Occlusions

In this section, we first describe our training procedure given an input image region centered on the object, then the run-time inference of the pose. Finally, we explain how we identify the input image region in practice.

10.4.1 Training

Datasets for 3D pose estimation typically provide training images annotated with the objects' poses and the 3D models of the objects. From this data, we generate our training set $\{(I^{(i)}, \{\mathbf{p}_j^{(i)}\}_j, M^{(i)})\}_i$, where $I^{(i)}$ is the i-th training image; $\mathbf{p}_j^{(i)}$, the 2D projection of the j-th 3D corner; and $M^{(i)}$, the 2D mask of the object in image $I^{(i)}$. This mask can be obtained by projecting the 3D object model into the image using the object's pose.

10.4.1.1 The Unambiguous Case

Let us first ignore the fact that some image patches can be ambiguous and that the learning problem is actually multimodal. We train a network $f_\theta(\cdot)$ to predict a heatmap for each projection $\mathbf{p}_j$. The architecture we use for this network is shown in Figure 10.7. $f_\theta(\cdot)$ takes an input patch of size 32×32, and predicts a set of heatmaps of size 128×128, and

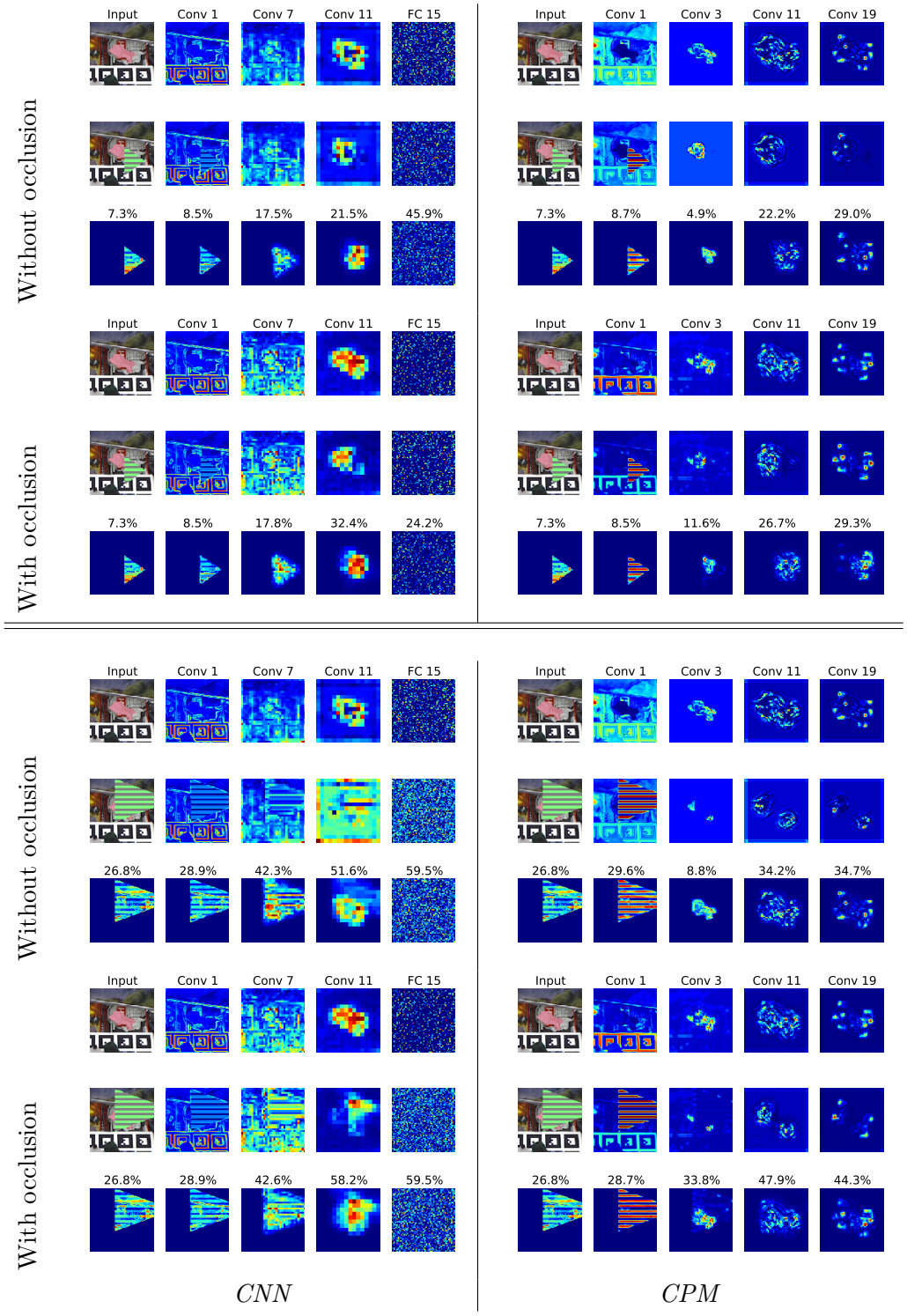


Figure 10.6: Effect of occlusions on the feature maps of *CNNs*. The left column shows the effects for a standard *CNN* and the right column for the *CPM*. We compare the influence on the feature maps when the network was trained with and without occlusions. The top row of each subfigure shows the feature maps for different layers of the network when no occlusion is present in the input image. The middle row shows the feature maps when an occlusion is present in the input image, and the bottom row show the absolute difference between the feature maps. We further denote the fraction of features where the difference is larger than 10%. Up to 60% of the features can be affected by the occlusions, specifically for standard *CNNs*. For the *CPM*, the influence of the occlusions is more localized to the occluded region of the input image.

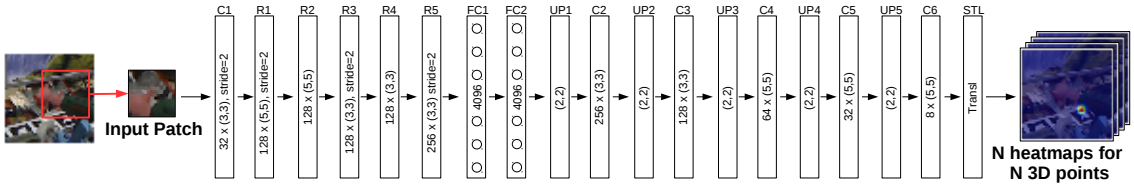


Figure 10.7: Network architecture for $f_\theta(\cdot)$. C denotes a convolutional layer with the number of filters and the filter size inscribed; FC, a fully connected layer with the number of neurons; UP, an unpooling layer [314]; R, a residual module [92] with the number of filters and filter size; and STL, a Spatial Transformation Layer [111] used for translation. All layers have ReLU activations, and the output of the last layer is linear.

we train it by minimizing:

$$\min_\theta \sum_i \sum_{u,v} \|\mathcal{H}^{(i)} - \text{Transl}(f_\theta(\mathcal{P}(I^{(i)}, u, v)), -u, -v)\|^2 , \tag{10.1}$$

where:

- $\mathcal{P}(I^{(i)}, u, v)$ is an image patch centered on location (u, v) in image $I^{(i)}$;
- $\mathcal{H}^{(i)}$ is the set of expected heatmaps for $\mathcal{P}(I^{(i)}, u, v)$. It contains one heatmap for each 2D projection $\mathbf{p}_j^{(i)}$. We describe how $\mathcal{H}^{(i)}$ is defined in detail below;
- $f_\theta(\cdot)$ returns a set of heatmaps, one for each 2D projection $\mathbf{p}_j^{(i)}$.
- $\text{Transl}(H, -u, -v)$ translates the predicted heatmaps H by $(-u, -v)$. $f_\theta(\cdot)$ learns to predict the heatmaps with respect to the patch center (u, v), and this translation is required to correctly align the predicted heatmaps together. Such a translation can be efficiently implemented using a Spatial Transformation Layer [111], which makes the network trainable end-to-end.

The sum $\sum_{(u,v)}$ is over 2D locations randomly sampled from the image. The heatmaps in $\mathcal{H}^{(i)}$ are defined as a Gaussian distribution with a small standard deviation (we use $\sigma = 4\,\text{px}$ in practice) and centered on the expected 2D projections $\mathbf{p}_j^{(i)}$ when patch $\mathcal{P}(I^{(i)}, u, v)$ overlaps the object mask $M^{(i)}$. The top row of Figure 10.4 shows examples of such heatmaps.

When the patch does not overlap the object mask, the heatmaps in $\mathcal{H}^{(i)}$ are defined as a uniform distribution of value $\frac{1}{W \cdot H}$, where $W \times H$ is the heatmap's resolution, since there is no information in the patch to predict the 2D projections. In addition, we use patches sampled from the ImageNet dataset [51] and train the network to predict uniform heatmaps as well for these patches. Considering these patches (outside the object's mask or from ImageNet) during training allows us to correctly handle patches appearing in the background or on the occluders and significantly reduces the number of false positives observed at run-time.

10.4.1.2 The Multimodal Case

Let us now consider the real problem, where the prediction is multimodal: Two image patches such as the ones shown in Figure 10.4(c) can be similar but extracted from different training images and, therefore, correspond to different expected heatmaps. In other words, in our training set, we can have values for samples i, i' and locations (u, v) and (u', v') such that $\mathcal{P}(I^{(i)}, u, v) \approx \mathcal{P}(I^{(i')}, u', v')$ and $\mathcal{H}^{(i)} \neq \mathcal{H}^{(i')}$.

It may seem as though, in this case, training given by Eqn. (10.1) would fail or need to be modified. *In fact, Eqn. (10.1) remains valid.* This is because we use the least-squares loss function: For image patches with similar appearances that correspond to different possible heatmaps, $f_\theta(\cdot)$ will learn to predict the average of these heatmaps, which is exactly what we want. The bottom row of Figure 10.4 shows such heatmaps. At run-time, because we will combine the contribution of multiple image patches, we will be able to resolve the ambiguities.

10.4.2 Run-Time Inference

At run-time, given an input image I, we extract patches from randomly selected locations from the input image and feed them into the predictor $f_\theta(\cdot)$. To combine the contributions of the different patches, we use a simple ensemble approach and average the predicted heatmaps for each 2D projection. We take the locations of the global maxima after averaging them as the final predictions for the 2D projections.

More formally, the final prediction $\widetilde{\mathbf{p}_j}$ for the 2D projection $\mathbf{p}_j$ is the location of the global maximum of $\sum_{u,v} \text{Transl}(f_\theta(\mathcal{P}(I, u, v)), -u, -v)[j]$, the sum of the heatmaps predicted for the j-th projection, translated such that these heatmaps align correctly. The sum is performed over randomly sampled patches. To compute the pose, we use a PnP estimation with RANSAC [90] on the correspondences between the corners of the object's 3D bounding box and the $\widetilde{\mathbf{p}_j}$ locations.

10.4.3 Two-Step Procedure

In practice, we first estimate the 2D location of the object of interest, using the same method as in the previous subsection, but instead of sampling random locations, we apply the network $f_\theta(\cdot)$ in a sliding window fashion, as illustrated in Figure 10.2b. For each image location, we compute a score by summing up the heatmap values over a bounding box of size 128×128 and over the 8 corners for each object, which is done efficiently using integral images. We apply Gaussian smoothing and thresholding to the resulting score map. We use the centers of mass of the regions after thresholding as the centers of the input image I. Finally, we use this image as input to the method described in the previous subsection. We use a fixed size for this region as our method is robust to scale changes.

10.5 Evaluation

In this section, we evaluate our method and compare it to the state-of-the-art. For this, we use two datasets: The Occluded LineMOD dataset [25] and the YCB-Video dataset [297]. Both datasets contain challenging sequences with partially occluded objects and cluttered backgrounds. In the following, we first provide the implementation details, the evaluation metrics used and then present the results of evaluation of the two datasets, including the results of an ablative analysis of our method.

10.5.1 Implementation Details

Training Data The training data consist of real and synthetic images with annotated 3D poses and object masks, as was also the case in [297]. To render the synthetic objects, we use the models that are provided with the datasets. We crop the objects of interest from the training images and paste them onto random backgrounds [58] sampled from ImageNet [51] to achieve invariance to different backgrounds. We augment the dataset with small affine perturbations in HSV color space.

Network Training The network is optimized using ADAM [125] with default parameters and using a minibatch size of 64, a learning rate of 0.001, and 100k iterations. We train one network per object starting from a random initialization.

Symmetric Objects We adapt the heatmap generation to symmetric objects present in the two datasets. For rotationally symmetric objects, *e.g.*, cylindrical shapes, we only predict a single position around the rotation axis. For mirror-symmetric objects, we only train on half the range of the symmetry axis, as was performed in [203].

Feature Mapping Optionally, we apply the Feature Mapping method as described in Chapter 7 to compensate for a lack of real training data. We apply the mapping between the FC1 and FC2 layers shown in Figure 10.7. The mapping network uses the same architecture as shown in Figure 7.4, but the weight for the feature loss is significantly lower (10^{-5}).

10.5.2 Evaluation Metrics

We consider the most common metrics. The 2D Reprojection error [26] computes the distances between the projections of the 3D model points when projected using the ground truth pose, and when using the predicted pose. The ADD metric [94] calculates the average distance in 3D between the model points, after applying the ground truth pose and the predicted pose. For symmetric objects, the 3D distances are calculated between the closest 3D points, denoted as the ADI metric. Below, we refer to these two metrics as AD{D|I} and use the one appropriate to the object.

10.5.3 Runtime

We implemented our method in Python on an Intel i7 with 3.2GHz and 64GB of RAM, using an nVidia GTX 980 Ti graphics card. Pose estimation is 100 ms for 64 patches, and detection takes 150 ms on a 640×480 camera frame. Predicting the heatmaps for a single patch takes 4 ms, and the total runtime could, thus, be significantly reduced by processing the individual patches in parallel.

10.5.4 Occluded LineMOD Dataset

We evaluate our approach on the Occluded LineMOD dataset [25], and we use *only the color images* for our method and all results reported.

For training the heatmap predictors, we use the LineMOD dataset [94] that contains the same objects as the Occluded LineMOD dataset. This protocol is commonly used for the dataset [112, 203, 264, 297], since the Occluded LineMOD dataset only contains testing sequences. Figure 10.8 shows some of the qualitative results obtained. We give an extensive quantitative evaluation in the following section.

10.5.4.1 Comparison with Baselines

Figure 10.9 shows the fraction of frames where the 2D Reprojection error is smaller than a given threshold, for each of the 8 objects from the dataset. A larger area under the curve denotes better results. We compare these results to those obtained from the use of several recent methods that also work only with color images, namely, BB8 [203], PoseCNN [297], Jafari *et al.* [112], and Tekin *et al.* [264]. Note that the method described in [203] uses ground truth detection, whereas ours does not. Our method performs significantly more accurately on all sequences. Notably, we also provide results for the *Eggbox* object, which, so far, was not considered since it was too difficult to learn for [112, 203, 264].

Adding Feature Mapping, which we introduced in Chapter 7, improves the 2D Reprojection error for a threshold of 5 px by 17% on average. We also tried Feature Mapping for the approach of [203], but it did not improve the results because the occlusions influence the feature maps too greatly when the network input contains occluders, as already discussed in the introduction.

Further quantitative results are given in Table 10.1, where we provide the percentage of frames for which the ADD or ADI metric is smaller than 10% of the object diameter, as [297] reported such results on the Occluded LineMOD dataset. This is considered a highly challenging metric. We also give the percentage of frames that have a 2D Reprojection error of less than 5 px. Our method significantly outperforms all other methods on these metrics by a large margin.

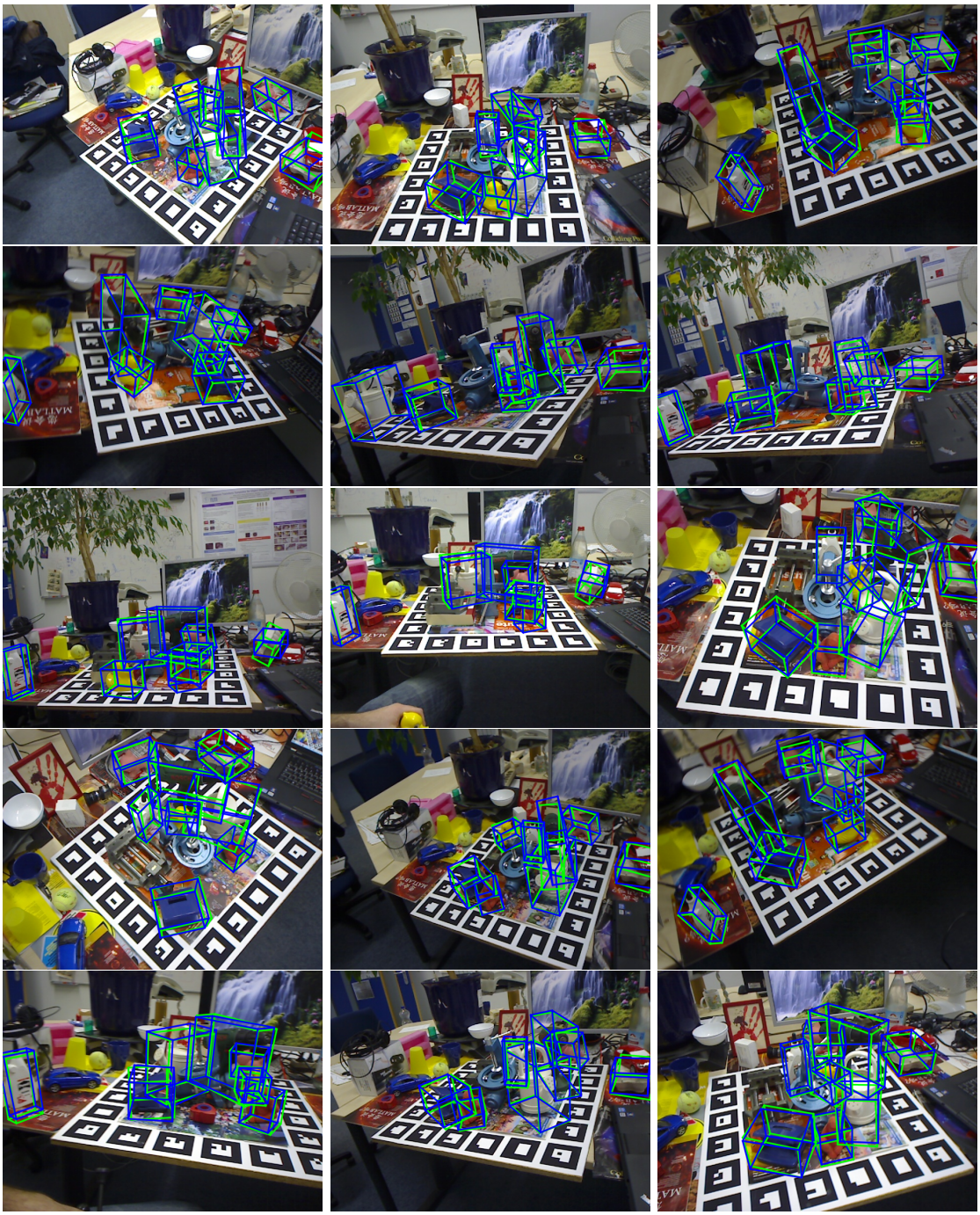


Figure 10.8: Some qualitative results on the Occluded LineMOD dataset [25]. We show the 3D bounding boxes of the objects projected onto the color image. Ground truth poses are shown in green, and our predictions are shown in blue.

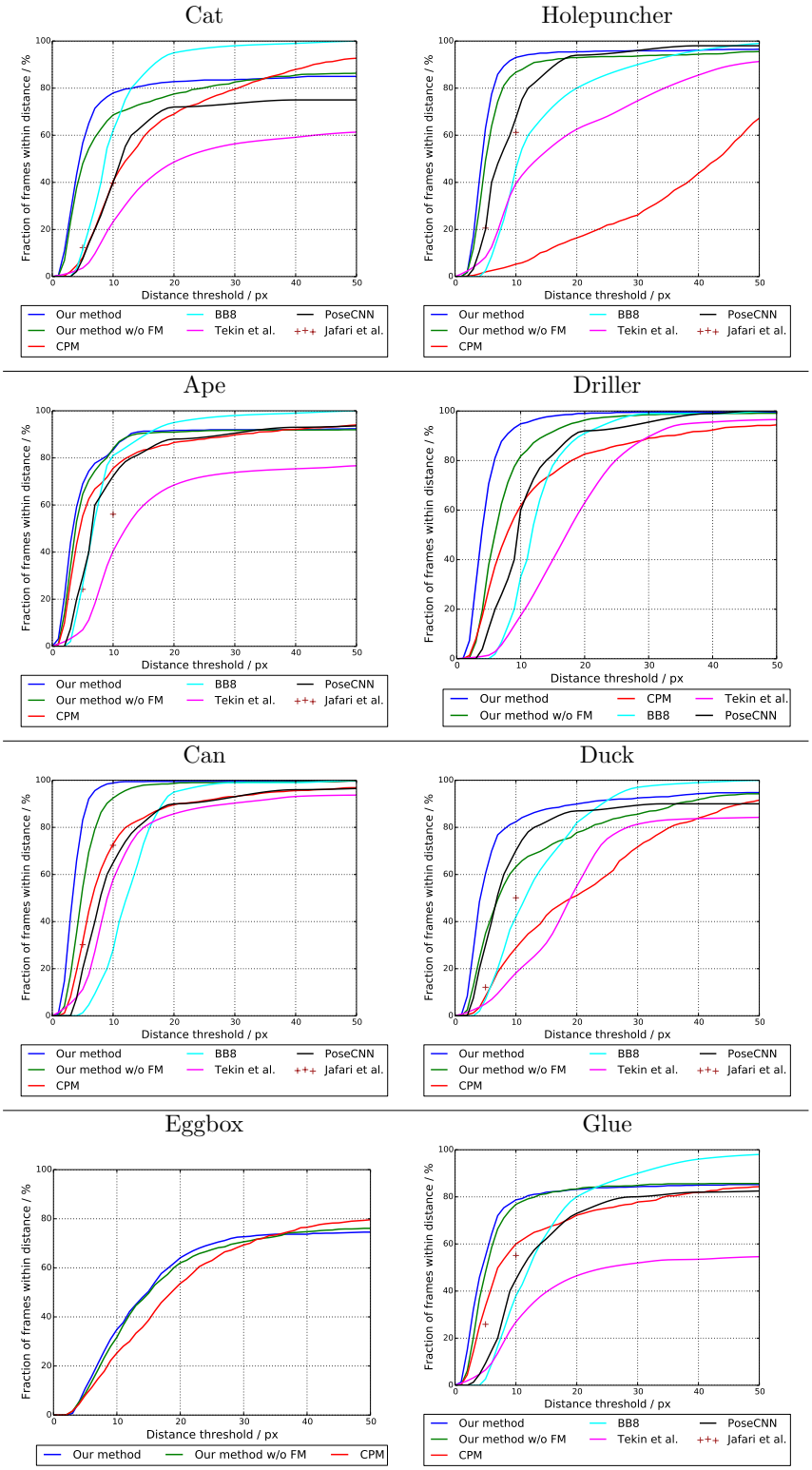


Figure 10.9: Evaluation on the Occluded LineMOD dataset [25] using color images only. We plot the fraction of frames for which the 2D Reprojection error is smaller than the threshold on the horizontal axis. Our method provides results that significantly outperform those reported by previous work. The suffix "w/o FM" denotes without Feature Mapping.

Method	AD{D|I}-10%									2D Reprojection Error-5 px								
	Ape	Can	Cat	Driller	Duck	Eggbox*	Glue*	Holepun.	Average	Ape	Can	Cat	Driller	Duck	Eggbox*	Glue*	Holepun.	Average
PoseCNN [297]	9.6	*45.2*	0.93	41.4	**19.6**	22.0	38.5	**22.1**	24.9	34.6	15.1	10.4	7.40	31.8	1.90	13.8	23.1	17.2
Tekin *et al.* [264]	–	–	–	–	–	–	–	–	–	7.01	11.2	3.62	1.40	5.07	–	6.53	8.26	6.16
BB8 [203]	–	–	–	–	–	–	–	–	–	28.5	1.20	9.60	0.00	6.80	–	4.70	2.40	7.60
Jafari *et al.* [112]	–	–	–	–	–	–	–	–	–	24.2	30.2	12.3	–	12.1	–	25.9	20.6	20.8
CPM [290]	12.5	25.6	1.43	23.8	6.99	18.3	15.0	0.74	13.0	55.4	30.6	15.7	27.9	26.6	7.97	18.5	1.81	23.1
Our method w/o FM	*16.5*	42.5	*2.82*	*47.1*	11.0	*24.7*	*39.5*	*21.9*	*25.8*	*64.7*	*53.0*	*47.9*	*35.1*	*36.1*	*10.3*	*44.9*	*52.9*	*43.1*
Our method	**17.6**	**53.9**	**3.31**	**62.4**	*19.2*	**25.9**	**39.6**	21.3	**30.4**	**69.6**	**82.6**	**65.1**	**73.8**	**61.4**	**13.1**	**54.9**	**66.4**	**60.9**

Table 10.1: Comparison on the Occluded LineMOD dataset [25] with color images only. We provide the percentage of frames for which the AD{D|I} error is smaller than 10% of the object diameter, and for which the 2D Reprojection error is smaller than 5 px. Objects marked with a * are considered to be symmetric. The suffix "w/o FM" denotes without Feature Mapping.

10.5.4.2 The Effect of Seeing Occlusions During Training

We evaluate the importance of knowing the occluder in advance. [203, 264, 297] assumed that the occluder is another object from the LineMOD dataset and only used occlusions from these objects during training. However, in practice, this assumption does not hold, since the occluder can be an arbitrary object. Therefore, we investigated how the performance was affected by the use of occlusions during training.

We compare our results (without Feature Mapping) to two state-of-the-art approaches: our reimplementations of BB8 [203] and *CPM* [290]. To avoid bias introduced by the limited amount of training data in the Occluded LineMOD dataset [25], we consider synthetic images both for training and for testing here.

We investigate three different training schemes: (a) No occlusions used for training; (b) random occlusions by simple geometric shapes; (c) random occlusions with the same objects from the dataset, as described in [112, 203, 264]. We compare the different training schemes in Figure 10.10. Training without occlusions clearly result in worse performance for BB8 and *CPM*, whereas our method is significantly more robust. Adding random geometric occlusions during training slightly increases the performance of BB8 and *CPM*, since the networks learn invariance to occlusions, however, mainly for these specific occlusions, whereas our approach maintains the accuracy compared to training without occlusions. Using occluders from the dataset gives the best results, since the networks learn to ignore specific features from these occluders. This, however, is only possible when the occluders are known in advance, which is not necessarily the case in practice.

10.5.4.3 Patch Size

We evaluated the influence of the patch size on the predicted pose accuracy. There is a range of sizes (25 px to 40 px) for which the performances stay very close to those presented in Table 10.1. Small patches seem to lack discriminative power, the 2D Reprojection metric gets 19% worse with 8 px patches, and large patches are sensitive to occlusions, which leads

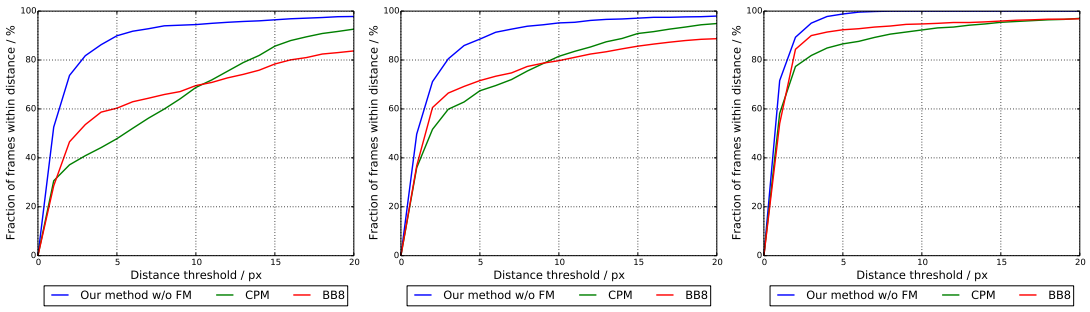


Figure 10.10: Evaluation of synthetic renderings of scenes from the Occluded LineMOD dataset (see text) using the 2D Reprojection error. (a) Training without occlusions; (b) training with random geometric occlusions; and (c) training with occluding objects from the LineMOD dataset [94]. Knowing the occluders in advance significantly improves the performances of BB8 [203] and *CPM* [290], however, this knowledge is often not available in practice. Our method does not require this knowledge.

to a decrease in the 2D Reprojection metric of 5% for 128 px patches.

10.5.4.4 Number of Patches

Figure 10.11 shows the aggregated heatmaps for one input image when varying the number of patches.

Figure 10.12a shows the influence of the sampling of the patches on the 2D Reprojection error for different sampling strategies. Since we do not have a segmentation mask of the object given during run-time, we rely on random sampling. However, if we sample enough patches, *i.e.*, more than 64 patches, we can achieve the same performance as sampling the patches only on the object mask that we created for the test sequences. If some preprocessing algorithm would provide a segmentation mask as well, we could reduce the number of patches to achieve the same accuracy. Figure 10.12b shows the influence of the number of patches that we randomly sample. The more patches we sample, the more accurate the results get, however, the accuracy starts to flatten for more than 64 patches.

10.5.5 YCB-Video Dataset

Further, we evaluate our approach an the recently proposed YCB-Video dataset [297]. Figure 10.13 shows some qualitative results. We give an extensive quantitative evaluation in the following section.

A detailed evaluation of the different objects of the YCB-Video dataset is shown in Table 10.2. Our method performs better for almost all objects and significantly better on all metrics on average. In [297], the area under the accuracy-threshold curve was used as

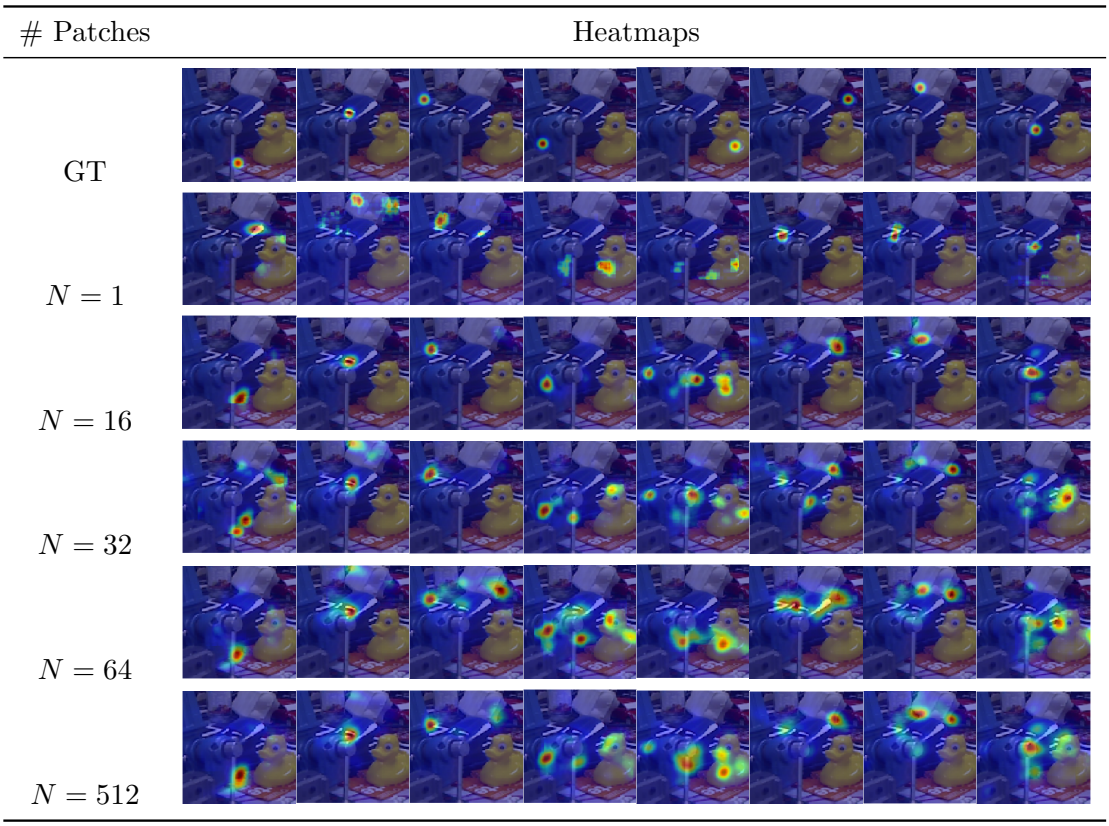


Figure 10.11: Effect of the number of patches used at run-time. While for small numbers of patches, the resulting heatmaps are strongly effected by the sampling, the aggregation of more heatmaps forms more accurate distributions robust to the patches sampled from occluded regions.

a metric, which we also provide.[1]

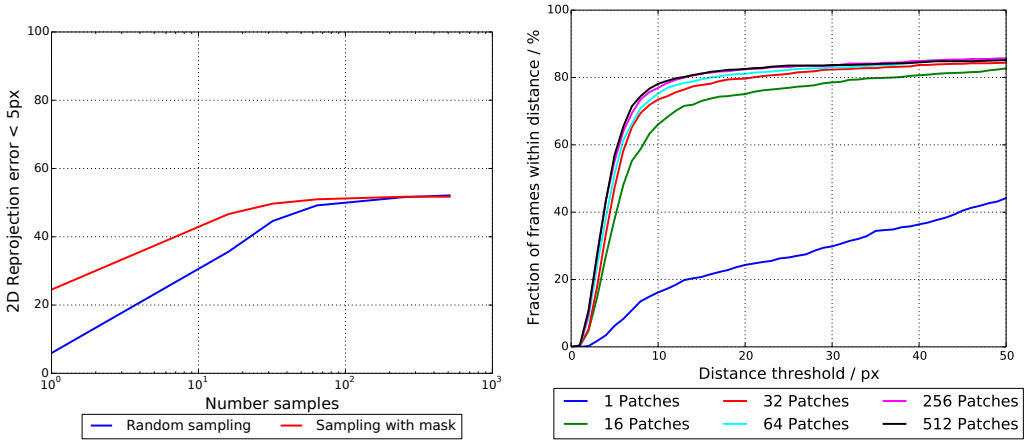
Figure 10.14a and 10.14b plot the fraction of frames where the 2D Reprojection error and the AD{D|I} metrics averaged over all the objects are smaller than a given threshold. Our approach clearly results in better performance.

A qualitative comparison of our proposed method to the state-of-the-art method PoseCNN [297] is shown in Figure 10.15. Our method provides significantly more accurate results compared to PoseCNN.

10.6 Discussion

In this chapter, we introduced a novel method for 3D object pose estimation that is inherently robust to partial occlusions of the object. To do this, we considered only small image patches as input and merged their contributions. Because we chose to compute the pose

[1]The metrics are calculated from the results provided by the authors at their website.



(a) Sampling patches within the object mask compared to random sampling.

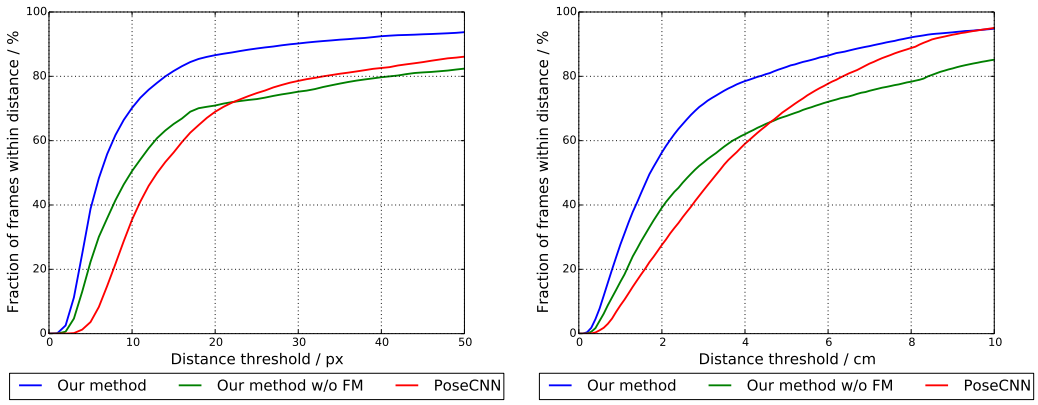
(b) Influence of number of samples.

Figure 10.12: (a) Evaluation of the sampling strategy and (b) the number of sampled patches on the Occluded LineMOD dataset [25] for the *Cat* object. We plot the fraction of frames where the 2D Reprojection error is smaller than a threshold.

Method / Object	PoseCNN [297] AUC	PoseCNN [297] AD{D|I}-10%	PoseCNN [297] 2D Repr-5 px	Our method w/o FM AUC	Our method w/o FM AD{D|I}-10%	Our method w/o FM 2D Repr-5 px	Our method AUC	Our method AD{D|I}-10%	Our method 2D Repr-5 px
002_master_chef_can	50.1	3.58	0.09	*68.5*	*32.9*	*9.94*	**81.6**	**75.8**	**29.7**
003_cracker_box	52.9	25.1	0.12	*74.7*	*62.6*	*24.5*	**83.6**	**86.2**	**64.7**
004_sugar_box	68.3	40.3	7.11	*74.9*	*44.5*	*47.0*	**82.0**	**67.7**	**72.2**
005_tomato_soup_can	66.1	25.5	5.21	*68.7*	*31.1*	**41.5**	**79.7**	**38.1**	*39.8*
006_mustard_bottle	*80.8*	*61.9*	6.44	72.6	42.0	*42.3*	**91.4**	**95.2**	**87.7**
007_tuna_fish_can	**70.6**	**11.4**	2.96	38.2	*6.79*	*7.14*	*49.2*	5.83	**38.9**
008_pudding_box	62.2	14.5	5.14	*82.9*	*58.4*	*43.9*	**90.1**	**82.2**	**78.0**
009_gelatin_box	74.8	12.1	15.8	*82.8*	*42.5*	*62.1*	**93.6**	**87.8**	**94.8**
010_potted_meat_can	59.5	18.9	23.1	*66.8*	*37.6*	*38.5*	**79.0**	**46.5**	**41.2**
011_banana	**72.1**	*30.3*	0.26	44.9	16.8	*8.18*	*51.9*	**30.8**	**10.3**
019_pitcher_base	53.1	15.6	0.00	**70.3**	*57.2*	**15.9**	*69.4*	**57.9**	*5.43*
021_bleach_cleanser	50.2	21.2	1.16	*67.1*	*65.3*	*12.1*	**76.1**	**73.3**	**23.2**
024_bowl*	*69.8*	12.1	4.43	58.6	*25.6*	*16.0*	**76.9**	**36.9**	**26.1**
025_mug	**58.4**	5.18	0.78	38.0	*11.6*	*20.3*	*53.7*	**17.5**	**29.2**
035_power_drill	55.2	29.9	3.31	*72.6*	*46.1*	*40.9*	**82.7**	**78.8**	**69.5**
036_wood_block*	**61.8**	10.7	0.00	*57.7*	**34.3**	**2.48**	55.0	*33.9*	*2.06*
037_scissors	*35.3*	*2.21*	0.00	30.9	0.00	0.00	**65.9**	**43.1**	**12.1**
040_large_marker	**58.1**	*3.39*	*1.38*	46.2	3.24	0.00	*56.4*	**8.88**	**1.85**
051_large_clamp*	*50.1*	*28.5*	*0.28*	42.4	10.8	0.00	**67.5**	**50.1**	**24.2**
052_extra_large_clamp*	46.5	19.6	*0.58*	*48.1*	*29.6*	0.00	**53.9**	**32.5**	**1.32**
061_foam_brick*	*85.9*	*54.5*	0.00	82.7	51.7	*52.4*	**89.0**	**66.3**	**75.0**
Average	61.0	21.3	3.72	*61.4*	*33.6*	*23.1*	**72.8**	**53.1**	**39.4**

Table 10.2: Comparison on the YCB-Video dataset [297]. We denote the area under the accuracy-threshold curve (AUC), the percentage of frames which have a 2D Reprojection error of less than 5 px, and the percentage of frames where the 3D ADD or ADI error is less than 10% of the object diameter. The objects marked with * are considered to be symmetric. Our method clearly outperforms the baseline. The suffix "w/o FM" denotes without Feature Mapping.

Figure 10.13: Qualitative results on the YCB-Video dataset [297]. The green bounding boxes correspond to the ground truth poses, the blue ones to our estimated poses.



(a) 2D Reprojection error. **(b)** AD{D|I} error.

Figure 10.14: Evaluation on the YCB-Video dataset [297]. We plot the fraction of frames where (a) the 2D Reprojection error and (b) AD{D|I} metrics are smaller than a threshold. The suffix "w/o FM" denotes without Feature Mapping.

by initially predicting the 2D projections of 3D points related to the object, the prediction can be performed in the form of 2D heatmaps. Since heatmaps are closely related to density functions, they can be conveniently applied to capture the ambiguities that arise when using small image patches as input. We showed that training a network to predict the heatmaps in the presence of such ambiguities is much simpler than it may sound. This resulted in a simple pipeline, which outperformed much more complex methods on two challenging datasets.

Our approach can be extended in different ways. The heatmaps could be merged in a way that is more robust to erroneous values than simple averaging. The pose could be estimated by considering the best local maxima rather than only the global maxima. Sampling only patches intersecting with the object mask, which could be predicted by a segmentation method, would limit the influence of occluders and background in the accumulated heatmaps even more. Predicting the heatmaps could be performed in parallel to reduce the runtime.

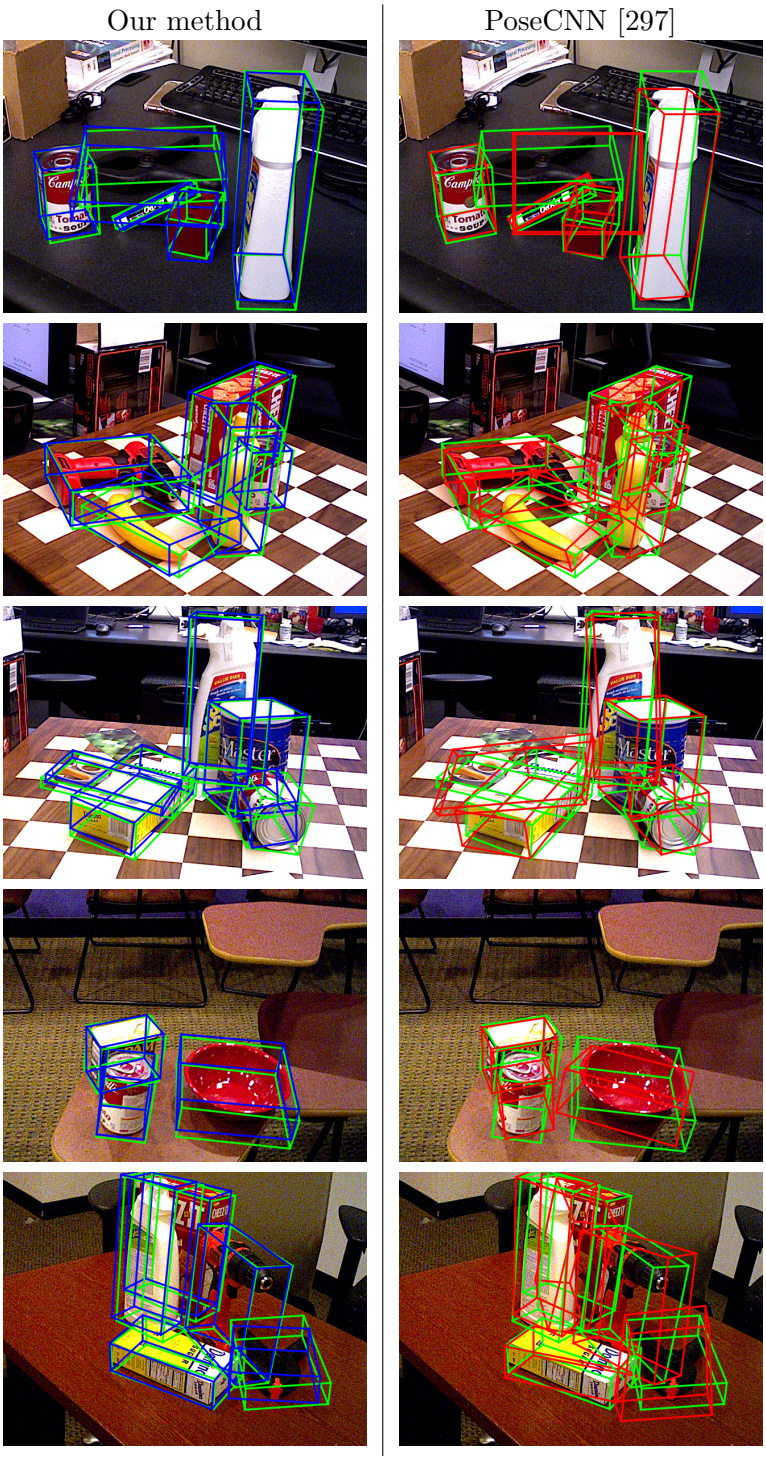


Figure 10.15: Qualitative comparison to state-of-the-art PoseCNN [297] on the YCB-Video dataset [297]. The green bounding boxes correspond to the ground truth poses, the blue ones to our estimated poses, and the red ones to the results from PoseCNN.

11 Conclusion

11.1 Summary

In this thesis we presented solutions to several important and unsolved problems in Computer Vision-based 3D hand pose estimation and joint hand-object pose estimation. More specifically, we introduced methods for fast and accurate 3D pose estimation in Part I. First, we showed in Chapter 4 that integrating a prior on the 3D hand pose can significantly increase the accuracy of the pose, without forfeit in computational efficiency and running at several hundred frames per second. Still, this estimator can make mistakes, and we showed in Chapter 5 how we can train an additional predictor to fix the mistakes made by an initial estimator.

In Part II, we proposed methods to deal with the limited training data available for 3D hand pose estimation. We introduced a method in Chapter 6 that can be used to generate dense 3D annotations for sequences showing a users hand from sparse 2D labels. We showed that acquiring these labels can be efficiently done in a semi-automatic way. Further, we showed in Chapter 7 that we can use synthetically generated depth images to boost the accuracy of the 3D pose estimator introduced in Chapter 4. We further showed in Chapter 8 that we can use pairs of real color and depth images as supervisory signal together with synthetically generated depth images to train an estimator for 3D hand pose estimation from color images without requiring labeled color images.

Finally, in Part III, we introduced methods for hand and object pose estimation. We first showed in Chapter 9 that we can generalize the method introduced in Chapter 5 to multiple objects, specifically estimating the 3D pose of the hand and a hand-held object jointly. Since occlusions of the object are very common when the hand is interacting with an object, we proposed a method for 3D pose estimation that is inherently robust to these occlusions.

In conclusion, we presented efficient and accurate methods for 3D hand pose estimation that are based on Deep Learning. Our methods do not require markers on the human hands. Also, they do not require temporal tracking since they are applied on each frame

separately, which makes them inherently robust to drift. Although we are very happy with the results from our methods, there is always room for improvement and we discuss possible future work in the next section.

11.2 Future Work

There are several directions for future work that naturally arise from the work presented herein. We state possible directions below. In general, there are several open challenges that need to be addressed in future work. Hand pose estimation needs to become more robust and should always result in plausible poses even under significant occlusion. Especially, when hands interact with hand-held objects and when hands are in contact with other parts of the environment. Hand pose estimation needs to work everywhere and for everyone. Also, it should work on inexpensive and low power sensor and devices.

Single Color Cameras For our pose estimation methods in Part I of this theses, we relied on a single depth camera for 3D hand pose estimation. However, depth sensors consume more energy than color cameras, have a limited range, and might not work in outdoor environments. In contrast, color cameras are already ubiquitous on different devices, such as smartphones or notebooks, and work in many different environments. In Chapter 8, we already presented a method to estimate the 3D pose from a single color image. Although we achieved encouraging results, there is still room for improvement. There is the essential question on how to lift the estimates from 2D color images to 3D. Also, clutter can be a bigger problem than for depth images since using simple heuristics for the segmentation, as we did in this thesis, do not work anymore. Finally, creating training data is more challenging, since using marker-based systems [88, 311], which are used so far for large scale datasets, does not work anymore as they would corrupt the images. We believe that 3D pose estimation from single color images is the most promising direction for future work especially for enabling 3D hand pose estimation at a large scale on mobile devices.

Person-Specific Pose Estimation In this thesis we relied upon an *ad hoc* method to normalize the hand size of different users, but this did not consider different hand shapes. The largest dataset we used contains ten users [311], and our experiments showed that the accuracy deteriorates for unknown users. There exists a large variability of hand shapes and sizes in practice [162], and when using color images as input even different skin colors. In terms of diversity, it is unclear how to deal with persons that have disabilities of the hands, or skin wrinkles of elderly persons [262], for example. Capturing datasets for such corner cases becomes quickly difficult and expensive, but might be necessary for widespread applicability in practice.
Adjusting the pose estimator to a specific user could be a possible solution [36], but so far there is no guarantee that this method improves the pose estimates in all cases, and

training on the users device quickly becomes impractical and intractable. A direction for further research could involve designing calibration procedures and integrating the acquired calibration into the pose estimator.

Hand-Object and Hand-Hand Pose Estimation As discussed in the previous paragraphs, creating pose estimation methods for hands in isolation can already be challenging. This becomes even harder when considering hands and objects jointly. Figure 11.1 shows example images and challenges that arise with this problem. In addition to self-occlusions of the hand, there can be severe occlusions from the object or other hands that make pose estimation very difficult. Not only pose inference is difficult in such cases, but also creating the training data in the first place. We would like to note that current datasets for this problem, which we also considered in Chapter 9, look rather artificial [173, 240]. There might be several further directions of creating training data within this context: Firstly, synthetic datasets could be used. However, it is hard to model the interactions between hands and objects, and additionally there is a domain gap between synthetic and real data that needs to be addressed. Secondly, multi-camera setups [241] could be used to mitigate the occlusion problem, but this would still require large efforts to automatize the annotation procedure in order to capture large datasets.

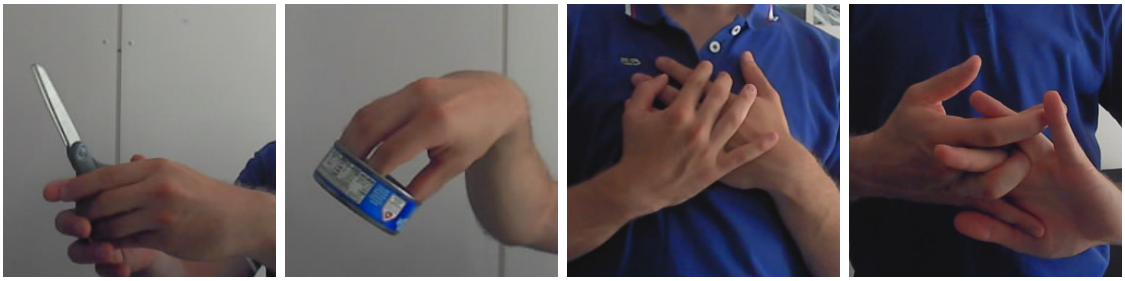


Figure 11.1: Challenges that arise in hand-object and hand-hand pose estimation. Not only parts of the hand are occluded by the object, but also parts of the object are occluded by the hand, or hands occlude each other or touch different body parts. This is challenging for both, the hand pose estimation and object pose estimation.

Tight Interactions In this thesis we have shown that 3D hand pose estimation already gives very good results for hands in isolation. However, when two hands are closely interacting, or a hand is grasping a object tightly, the accuracy of our methods quickly deteriorates. Although we have shown accurate results for joint hand-object pose estimation, the benchmark dataset used for the evaluation did not contain very tight interactions. While in Chapter 10 we designed a method for object pose estimation that is robust to unknown occluders, such properties are also desirable for hand pose estimation. Otherwise we would need to know all possible occluding objects in advance. However, the method we proposed relies on patches and can be confused with the self-similarity of the different

fingers. Thus, adding some kind of structural constraints to the predicted heatmaps that preserves the kinematic constraints of valid hand poses could be a possible solution. Also, adding physical constraints to the predictions could be a solution, as we discuss below.

Tight Integration in Graphics Engines For Augmented Reality (AR) and Virtual Reality (VR) applications, the visual feedback and interactions for the user are created using a graphics engine that renders the virtual scene. For physically correct scenes, the graphics engine takes the hand pose and performs physics simulation based on the hand pose in order to correctly arrange the scene objects. However, the hand pose is taken as an input to the engine and treated as a *correct* observation, not taking into account uncertainties of the measurements. This can lead to jitter, drift, or other uncomfortable glitches that deteriorate an immersive user experience. A possible direction could be to integrate the hand pose estimation method more tightly into the graphics engine and physics simulation, or even learning the full graphics pipeline with physical interaction in an end-to-end framework.

Physically Correct Interaction In the real world there are strong physical constraints for hand-object interaction, *e.g.*, interpenetration, friction, or drag. Figure 11.2 shows images of our work [98], where we modeled the friction forces between hands and objects in a graphics engine assuming a given 3D hand pose. In this thesis, we implicitly learned such constraints from the training data, but they were not strictly enforced during inference. However, such constraints can be a strong prior for the hand and the object pose. So far, it is unclear, how exactly this can be done. A possible direction is the work of [159], who introduced a method to enforce hard constraints in learning-based methods that could be added as an additional constraint layer to the joint feedback loop presented in Chapter 9.

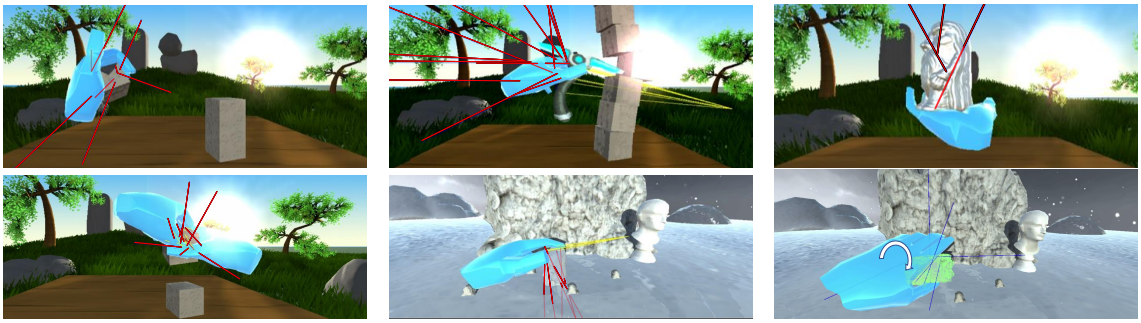


Figure 11.2: Physically correct interaction, taken from [98]. In [98], we aimed at modeling the interaction of a real hand with a virtual object by modeling the forces between the hand and the object with a simple friction model. However, we assumed that the 3D hand pose is given and correct, and the approach required a cumbersome integration into an off-the-shelf physics engine. The red arrows indicate the calculated friction forces that are applied to the objects. Yellow arrows depict the global hand motion.

Combined Discriminative and Generative Methods Combining discriminative methods with a generative model could combine the benefits from both worlds, *i.e.*, leading to hand pose estimators with the high accuracy of generative methods and the robustness to drift of discriminative methods. Such a framework was introduced in [217, 227], where a discriminative method can reinitialize the pose when tracking errors occur. However, the discriminative and generative parts are separated from each other and solely the hand pose serves as interface and is passed between the two parts. The feedback method presented in Chapter 5 could be reformulated as an energy minimization framework that performs the generative optimization of the hand pose.

Designing Efficient Human-Computer Interaction (HCI) Methods While we mainly focused on 3D hand pose estimation in this thesis, we skipped an important part: How can we leverage the accurate 3D pose for *HCI* methods? So far, most interaction methods are based on gestures. While gesture-based interaction tries to classify a hand movement into a certain gesture to trigger a function, our highly accurate 3D pose estimates would allow for much finer-grained interactions. For example, one might think of subtle manipulations of virtual objects. Such techniques could lead to a desirable increase in usability and user experience [238]. However, this would require to design such interaction methods and make them available in commodity interaction frameworks.

A

List of Acronyms

AAM	Active Appearance Model
AR	Augmented Reality
CNN	Convolutional Neural Network
CPM	Convolutional Pose Machine
DoF	Degrees of Freedom
EPE	End Point Error
fps	frames per second
GAN	Generative Adversarial Network
HCI	Human-Computer Interaction
ICP	Iterative Closest Point
ILP	Integer Linear Program
ISTN	Inverse Spatial Transformer Network
LBS	Linear Blend Skinning
MMD	Maximum Mean Discrepancy
PCA	Principal Component Analysis
PCK	Percentage of Correct Keypoints
PS	Pictorial Structures
PSO	Particle Swarm Optimization
RKHS	Reproducing Kernel Hilbert Space
STN	Spatial Transformer Network
VR	Virtual Reality

Bibliography

[1] Akiyama, K., Yang, F., and Wu, Y. (2017). 2017 HANDS Challenge: NAIST RVLab G2. (page 65)

[2] Albrecht, I., Haber, J., and Seidel, H.-P. (2003). Construction and Animation of Anatomically Based Human Hand Models. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. (page 2)

[3] Alhaija, H. A., Mustikovela, S. K., Mescheder, L., Geiger, A., and Rother, C. (2017). Augmented Reality Meets Deep Learning for Car Instance Segmentation in Urban Scenes. In *British Machine Vision Conference*. (page 118)

[4] Ali, K., Hasler, D., and Fleuret, F. (2011). Flowboost – Appearance Learning from Sparsely Annotated Data. In *Conference on Computer Vision and Pattern Recognition*. (page 99)

[5] Alugupally, N., Samal, A., Marx, D., and Bhatia, S. (2011). Analysis of Landmarks in Recognition of Face Expressions. *Pattern Recognition and Image Analysis*, 21(4):681–693. (page 19)

[6] Andriluka, M., Roth, S., and Schiele, B. (2009). Pictorial Structures Revisited: People Detection and Articulated Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[7] Athitsos, V. and Sclaroff, S. (2001a). 3D Hand Pose Estimation by Finding Appearance-Based Matches in a Large Database of Training Views. In *Workshop on Cues in Communication*. (page 5, 27)

[8] Athitsos, V. and Sclaroff, S. (2001b). Estimating 3D Hand Pose from a Cluttered Image. In *Conference on Computer Vision and Pattern Recognition*. (page 27)

[9] Babic, T., Reiterer, H., and Haller, M. (2017). GestureDrawer: One-handed Interaction Technique for Spatial User-defined Imaginary Interfaces. In *Symposium on Spatial User Interaction*. (page 4)

[10] Bailly, G., Elisei, F., Badin, P., and Savariaux, C. (2006). Degrees of Freedom of Facial Movements in Face-to-Face Conversational Speech. In *International Workshop on Multimodal Corpora*. (page 19)

[11] Baktashmotlagh, M., Harandi, M., Lovell, B., and Salzmann, M. (2013). Unsupervised Domain Adaptation by Domain Invariant Projection. In *International Conference on Computer Vision*. (page 118, 120)

[12] Ballan, L., Taneja, A., Gall, J., Van Gool, L., and Pollefeys, M. (2012). Motion Capture of Hands in Action Using Discriminative Salient Points. In *European Conference on Computer Vision*. (page 25, 26, 69, 96, 98)

[13] Belagiannis, V., Amin, S., Andriluka, M., Schiele, B., Navab, N., and Ilic, S. (2014). 3D Pictorial Structures for Multiple Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[14] Belagiannis, V., Rupprecht, C., Carneiro, G., and Navab, N. (2015). Robust Optimization for Deep Regression. In *International Conference on Computer Vision*. (page 96)

[15] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A CPU and GPU Math Expression Compiler. In *SciPy*. (page 57, 89, 130, 143, 156)

[16] Berkelaar, M., Eikland, K., and Notebaert, P. (2005). *Open Source (Mixed-Integer) Linear Programming System*. (page 100)

[17] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer. (page 71)

[18] Blanz, V. and Vetter, T. (1999). A Morphable Model for the Synthesis of 3D Faces. In *ACM SIGGRAPH*. (page 20)

[19] Blascovich, J. and Bailenson, J. (2011). *Infinite Reality: Avatars, Eternal Life, New Worlds, and the Dawn of the Virtual Revolution*. William Morrow & Co. (page 4)

[20] Bogo, F., Kanazawa, A., Lassner, C., Gehler, P., Romero, J., and Black, M. J. (2016). Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image. In *European Conference on Computer Vision*. (page 23)

[21] Boonbrahm, P. and Kaewrat, C. (2014). Assembly of the Virtual Model with Real Hands Using Augmented Reality Technology. In *International Conference on Virtual, Augmented and Mixed Reality*. (page 4)

[22] Bouchacourt, D., Kumar, M. P., and Nowozin, S. (2016). DISCO Nets: Dissimilarity Coefficient Networks. In *Advances in Neural Information Processing Systems*. (page 29, 59, 84, 127, 128)

[23] Bourdev, L. and Malik, J. (2009). Poselets: Body Part Detectors Trained Using 3D Human Pose Annotations. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[24] Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D., and Erhan, D. (2016). Domain Separation Networks. In *Advances in Neural Information Processing Systems*. (page 119, 121, 135, 136)

[25] Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). Learning 6D Object Pose Estimation using 3D Object Coordinates. In *European Conference on Computer Vision*. (page 30, 37, 38, 162, 163, 165, 170, 171, 172, 173, 174, 177)

[26] Brachmann, E., Michel, F., Krull, A., Yang, M. M., Gumhold, S., and Rother, C. (2016). Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image. In *Conference on Computer Vision and Pattern Recognition*. (page 41, 160, 163, 170)

[27] Briggs, F. and Usrey, W. (2008). Emerging Views of Corticothalamic Function. *Current Opinion in Neurobiology*, 18(4):403–7. (page 90)

[28] Buch, A. G., Kiforenko, L., and Kraft, D. (2017). Rotational Subgroup Voting and Pose Clustering for Robust 3D Object Recognition. In *International Conference on Computer Vision*. (page 164)

[29] Busto, P. P. and Gall, J. (2017). Open Set Domain Adaption. In *International Conference on Computer Vision*. (page 121)

[30] Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5). (page 84)

[31] Cai, G., Wang, Y., Zhou, M., and He, L. (2018). Unsupervised Domain Adaptation with Adversarial Residual Transform Networks. In *arXiv Preprint*. (page 136, 137)

[32] Calli, B., Singh, A., Bruce, J., Walsman, A., Konolige, K., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2017). Yale-CMU-Berkeley Dataset for Robotic Manipulation Research. *International Journal of Robotics Research*, 36:261–268. (page 38, 136, 160)

[33] Cao, X., Wei, Y., Wen, F., and Sun, J. (2014). Face Alignment by Explicit Shape Regression. *International Journal of Computer Vision*, 107(2):177–190. (page 20)

[34] Carreira, J., Agrawal, P., Fragkiadaki, K., and Malik, J. (2016). Human Pose Estimation with Iterative Error Feedback. In *Conference on Computer Vision and Pattern Recognition*. (page 71)

[35] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. Technical report, Stanford University — Princeton University — Toyota Technological Institute at Chicago. (page 119, 153)

[36] Charles, J., Pfister, T., Magee, D., Hogg, D., and Zisserman, A. (2016). Personalizing Human Video Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 182)

[37] Chen, L., Wei, H., and Ferryman, J. (2013). A Survey of Human Motion Analysis using Depth Imagery. *Pattern Recognition Letters*, 34(15):1995–2006. (page 21)

[38] Chen, T., Wu, M., Hsieh, Y., and Fu, L. (2016). Deep Learning for Integrated Hand Detection and Pose Estimation. In *International Conference on Pattern Recognition*. (page 30)

[39] Chen, X., Wang, G., Guo, H., and Zhang, C. (2018). Pose Guided Structured Region Ensemble Network for Cascaded Hand Pose Estimation. *Neurocomputing*. (page 29, 65, 84)

[40] Choi, C., Sinha, A., Choi, J. H., Jang, S., and Ramani, K. (2015). A Collaborative Filtering Approach to Real-Time Hand Pose Estimation. In *International Conference on Computer Vision*. (page 28)

[41] Choo, B., Landau, M., DeVore, M., and Beling, P. A. (2014). Statistical Analysis-Based Error Models for the Microsoft Kinect Depth Sensor. *Sensors*, 2014(14):17430–17450. (page 7)

[42] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *Conference on Computer Vision and Pattern Recognition*. (page 77)

[43] Coates, A., Ng, A. Y., and Lee, H. (2011). An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *International Conference on Artificial Intelligence and Statistics*. (page 46)

[44] Cootes, T., Edwards, G., and Taylor, C. (1998). Active Appearance Models. In *European Conference on Computer Vision*. (page 20)

[45] Crivellaro, A., Rad, M., Verdie, Y., Yi, K., Fua, P., and Lepetit, V. (2017). Robust 3D Object Tracking from Monocular Images Using Stable Parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1465–1479. (page 163, 164)

[46] Crivellaro, A., Rad, M., Verdie, Y., Yi, K. M., Fua, P., and Lepetit, V. (2015). A Novel Representation of Parts for Accurate 3D Object Detection and Tracking in Monocular Images. In *International Conference on Computer Vision*. (page 30)

[47] Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Conference on Computer Vision and Pattern Recognition*. (page 100)

[48] Dantone, M., Gall, J., Leistner, C., and Van Gool, L. (2013). Human Pose Estimation using Body Parts Dependent Joint Regressors. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[49] Dayan, P. and Abbott, L. (2005). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press. (page 90)

[50] de La Gorce, M., Fleet, D. J., and Paragios, N. (2011). Model-Based 3D Hand Pose Estimation from Monocular Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1793–1805. (page 10, 25, 26, 27, 69)

[51] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *Conference on Computer Vision and Pattern Recognition*. (page 168, 170)

[52] Deng, X., Yang, S., Zhang, Y., Tan, P., Chang, L., and Wang, H. (2017). Hand3D: Hand Pose Estimation Using 3D Neural Network. In *arXiv Preprint*. (page 10, 28, 29, 59, 61, 84, 127, 128)

[53] Dipietro, L., Sabatini, A. M., and Dario, P. (2008). A Survey of Glove-Based Systems and Their Applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(4):461–482. (page 10)

[54] Doliotis, P., Athitsos, V., Kosmopoulos, D., and Perantonis, S. (2012). Hand Shape and 3D Pose Estimation Using Depth Data from a Single Cluttered Frame. In *Advances in Visual Computing*. (page 27)

[55] Dosovitskiy, A., Springenberg, J. T., and Brox, T. (2015). Learning to Generate Chairs with Convolutional Neural Networks. In *Conference on Computer Vision and Pattern Recognition*. (page 70, 71, 76)

[56] Doumanoglou, A., Balntas, V., Kouskouridas, R., and Kim, T. (2016). Siamese Regression Networks with Efficient Mid-Level Feature Extraction for 3D Object Pose Estimation. In *arXiv Preprint*. (page 164)

[57] Drost, B., Ulrich, M., Navab, N., and Ilic, S. (2010). Model Globally, Match Locally: Efficient and Robust 3D Object Recognition. In *Conference on Computer Vision and Pattern Recognition*. (page 30)

[58] Dwibedi, D., Misra, I., and Hebert, M. (2017). Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection. In *International Conference on Computer Vision*. (page 170)

[59] Edwards, G., Lanitis, A., Taylor, C., and Cootes, T. (1998). Statistical Models of Face Images – Improving Specificity. *Image and Vision Computing*, 16(3):203–211. (page 20)

[60] Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth Map Prediction from a Single Image Using a Multi-Scale Deep Network. In *Advances in Neural Information Processing Systems*. (page 140, 142)

[61] Erol, A., Bebis, G., Nicolescu, M., Boyle, R. D., and Twombly, X. (2007). Vision-Based Hand Pose Estimation: A Review. *Computer Vision and Image Understanding*, 108(1-2):52–73. (page 8, 23)

[62] Fang, K., Bai, Y., Hinterstoisser, S., and Kalakrishnan, M. (2018). Multi-task Domain Adaptation for Deep Learning of Instance Grasping from Simulation. In *International Conference on Robotics and Automation*. (page 121)

[63] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning Hierarchical Features for Scene Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929. (page 47, 48, 57)

[64] Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial Structures for Object Recognition. *International Journal of Computer Vision*, 61(1):55–79. (page 22)

[65] Ferrari, V., Marin-Jimenez, M., and Zisserman, A. (2008). Progressive Search Space Reduction for Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[66] Ferri, F., Campione, G., Volta, R. D., Gianelli, C., and Gentilucci, M. (2011). Social Requests and Social Affordances: How they Affect the Kinematics of Motor Sequences during Interactions between Conspecifics. *PlOS ONE*, 6(1):1–9. (page 4)

[67] Fourure, D., Emonet, R., Fromont, E., Muselet, D., Neverova, N., Tremeau, A., and Wolf, C. (2017). Multi-Task, Multi-Domain Learning: Application to Semantic Segmentation and Pose Regression. *Neurocomputing*, 1(251):68–80. (page 29, 59, 61)

[68] Gall, J. and Lempitsky, V. (2009). Class-Specific Hough Forests for Object Detection. In *Conference on Computer Vision and Pattern Recognition*. (page 165)

[69] Ganin, Y. and Lempitsky, V. (2015). Unsupervised Domain Adaption by Backpropagation. In *International Conference on Machine Learning*. (page 120, 126, 127, 135, 136, 143)

[70] Garrido, P., Valgaerts, L., Rehmsen, O., Thormählen, T., Pérez, P., and Theobalt, C. (2014). Automatic Face Reenactment. In *Conference on Computer Vision and Pattern Recognition*. (page 19)

[71] Gavrila, D. M. and Davis, L. (1996). 3D Model-Based Tracking of Humans in Action: A Multi-View Approach. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[72] Ge, L., Liang, H., Yuan, J., and Thalmann, D. (2016). Robust 3D Hand Pose Estimation in Single Depth Images: From Single-View CNN to Multi-View CNNs. In *Conference on Computer Vision and Pattern Recognition*. (page 28, 36, 62)

[73] Ge, L., Liang, H., Yuan, J., and Thalmann, D. (2018a). Real-time 3D Hand Pose Estimation with 3D Convolutional Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. (page 7, 8, 29, 64, 65, 68, 84)

[74] Ge, L., Ren, Z., and Yuan, J. (2018b). Point-to-Point Regression PointNet for 3D Hand Pose Estimation. In *European Conference on Computer Vision*. (page 68)

[75] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition*. (page 120)

[76] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*. (page 71, 76, 119, 121, 135, 136)

[77] Google (2018). Tilt Brush by Google. `https://www.tiltbrush.com/`. Accessed: 2018-08-08. (page 3, 4)

[78] Gordon, C. C., Blackwell, C. L., Bradtmiller, B., Parham, J. L., Barrientos, P., Paquette, S. P., Corner, B. D., Carson, J. M., Venezia, J. C., Rockwell, B. M., Mucher, M., and Kristensen, S. (2012). 2012 Anthropometric Survey of US Army Personell: Methods and Summary Statistics. Technical Report NATICK/TR-15/007, U.S. Army Natick Soldier Research, Development and Engineering Center Natick, Massachusetts 01760-2642. (page 7)

[79] Goudie, D. and Galata, A. (2017). 3D Hand-Object Pose Estimation from Depth with Convolutional Neural Networks. In *Automated Face and Gesture Recognition*. (page 30, 31)

[80] Gretton, A., Borgwardt, K., Rasch, M. J., Schölkopf, B., and Smola, A. J. (2006). A Kernel Method for the Two-Sample Problem. In *Advances in Neural Information Processing Systems*. (page 120, 135, 137, 139)

[81] Gu, X., Zhang, Y., Sun, W., Bian, Y., Zhou, D., and Kristensson, P. O. (2016). Dexmo: An Inexpensive and Lightweight Mechanical Exoskeleton for Motion Capture and Force Feedback in VR. In *ACM Conference on Human Factors in Computing Systems*. (page 10)

[82] Guo, H., Wang, G., Chen, X., Zhang, C., Qiao, F., and Yang, H. (2017). Region Ensemble Network: Improving Convolutional Network for Hand Pose Estimation. In *International Conference on Image Processing*. (page 10, 28, 29, 59, 60, 84, 127, 128)

[83] Gupta, A., Vedaldi, A., and Zisserman, A. (2016a). Synthetic Data for Text Localisation in Natural Images. In *Conference on Computer Vision and Pattern Recognition*. (page 118, 137)

[84] Gupta, S., Arbelaez, P., Girshick, R., and Malik, J. (2015). Aligning 3D Models to RGB-D Images of Cluttered Scenes. In *Conference on Computer Vision and Pattern Recognition*. (page 30)

[85] Gupta, S., Hoffman, J., and Malik, J. (2016b). Cross Modal Distillation for Supervision Transfer. In *Conference on Computer Vision and Pattern Recognition*. (page 136)

[86] Gustus, A., Stillfried, G., Visser, J., Jörntell, H., and van der Smagt, P. (2012). Human Hand Modelling: Kinematics, Dynamics, Applications. *Biological Cybernetics*, 106(11):741–755. (page 3)

[87] Hamer, H., Schindler, K., Koller-Meier, E., and Van Gool, L. (2009). Tracking a Hand Manipulating an Object. In *International Conference on Computer Vision*. (page 30, 31)

[88] Han, S., Liu, B., Wang, R., Ye, Y., Twigg, C. D., and Kin, K. (2018). Online Optical Marker-based Hand Tracking with Deep Labels. *ACM Transactions on Graphics*, 37(4):166:1–166:10. (page 10, 134, 153, 182)

[89] Harris, C. and Stennett, C. (1990). RAPID - A Video Rate Object Tracker. In *British Machine Vision Conference*. (page 163)

[90] Hartley, R. and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision*. Cambridge University Press. (page 160, 163, 169)

[91] He, K., Zhang, X., Ren, S., and Sun, J. (2014). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *European Conference on Computer Vision*. (page 46)

[92] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Conference on Computer Vision and Pattern Recognition*. (page 51, 125, 139, 140, 168)

[93] Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V. (2011). Multimodal Templates for Real-Time Detection of Texture-Less Objects in Heavily Cluttered Scenes. In *International Conference on Computer Vision*. (page 100)

[94] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012). Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. In *Asian Conference on Computer Vision*. (page 41, 118, 170, 171, 175)

[95] Hinterstoisser, S., Lepetit, V., Wohlhart, P., and Konolige, K. (2017). On Pre-Trained Image Features and Synthetic Images for Deep Learning. In *arXiv Preprint*. (page 119, 120, 136)

[96] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507. (page 100)

[97] Hoffman, J., Gupta, S., Leong, J., Guadarrama, S., and Darrell, T. (2016). Cross-Modal Adaptation for RGB-D Detection. In *International Conference on Robotics and Automation*. (page 136, 137)

[98] Höll, M., Oberweger, M., Arth, C., and Lepetit, V. (2018). Efficient Physics-Based Implementation for Realistic Hand-Object Interaction in Virtual Reality. In *IEEE Conference on Virtual Reality and 3D User Interfaces*. (page 4, 184)

[99] Holz, C., Ofek, E., Sinclair, M., Strasnick, E., and Benko, H. (2018). Haptic Links: Bimanual Haptics for Virtual Reality Using Variable Stiffness Actuation. In *ACM Conference on Human Factors in Computing Systems*. (page 4)

[100] Honari, S., Molchanov, P., Tyree, S., Vincent, P., Pal, C., and Kautz, J. (2018). Improving Landmark Localization with Semi-Supervised Learning. In *Conference on Computer Vision and Pattern Recognition*. (page 65)

[101] HTC (2018). HTC Vive. `https://www.vive.com/eu/`. Accessed: 2018-08-08. (page 10)

[102] Hu, J., Lu, J., and Tan, Y.-P. (2015). Deep Transfer Metric Learning. In *Conference on Computer Vision and Pattern Recognition*. (page 120)

[103] Huang, X., Peng, Y., and Yuan, M. (2017). Cross-modal Common Representation Learning by Hybrid Transfer Network. In *International Joint Conference on Artificial Intelligence*. (page 137)

[104] Huber, P. J. (1964). Robust Estimation of a Location Parameter. *Annals of Mathematical Statistics*, 35(1):73–101. (page 53)

[105] Hueting, M., Reddy, P., Kim, V., Carr, N., Yumer, E., and Mitra, N. (2018). SeeThrough: Finding Chairs in Heavily Occluded Indoor Scene Images. In *International Conference on 3D Vision*. (page 164)

[106] Illade-Quinteiro, J., Brea, V. M., López, P., Cabello, D., and Doménech-Asensi, G. (2015). Distance Measurement Error in Time-of-Flight Sensors Due to Shot Noise. *Sensors*, 2015(15):4624–4642. (page 7)

[107] Ionescu, C., Carreira, J., and Sminchisescu, C. (2014). Iterated Second-Order Label Sensitive Pooling for 3D Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 22, 95, 96)

[108] Ishibuchi, K., Takemura, H., and Kishino, F. (1993). Real Time Hand Gesture Recognition using 3D Prediction Model. In *Systems Man and Cybernetics Conference*. (page 1, 23)

[109] Ishikawa, Y., Shizuki, B., and Hoshino, J. (2017). Evaluation of Finger Position Estimation with a Small Ranging Sensor Array. In *Symposium on Spatial User Interaction*. (page 3, 4)

[110] Ivekovic, S., Trucco, E., and Petillot, Y. R. (2008). Human Body Pose Estimation with Particle Swarm Optimisation. *Evolutionary Computation*, 16(4):509–528. (page 23)

[111] Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial Transformer Networks. In *Advances in Neural Information Processing Systems*. (page 149, 151, 168)

[112] Jafari, O. H., Mustikovela, S. K., Pertsch, K., Brachmann, E., and Rother, C. (2017). iPose: Instance-Aware 6D Pose Estimation of Partly Occluded Objects. In *arXiv Preprint*. (page 160, 163, 171, 174)

[113] Jain, A., Tompson, J., Andriluka, M., Taylor, G. W., and Bregler, C. (2014). Learning Human Pose Estimation Features with Convolutional Networks. In *International Conference for Learning Representations*. (page 22, 77)

[114] Ji, Y., Li, H., Yang, Y., and Li, S. (2017). Hierarchical Topology based Hand Pose Estimation from a Single Depth Image. *Multimedia Tools and Applications*, 4(1):1–16. (page 29)

[115] Johnson, S. and Everingham, M. (2010). Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation. In *British Machine Vision Conference*. (page 22)

[116] Joo, H., Simon, T., Li, X., Liu, H., Tan, L., Gui, L., Banerjee, S., Godisart, T., Nabbe, B., Matthews, I., Kanade, T., Nobuhara, S., and Sheikh, Y. (2015). Panoptic Studio: A Massively Multiview System for Social Interaction Capture. In *International Conference on Computer Vision*. (page 4, 5)

[117] Kabsch, W. (1976). A Solution for the Best Rotation to Relate Two Sets of Vectors. In *Acta Crystallographica*. (page 152)

[118] Kanazawa, A., Black, M. J., Jacobs, D. W., and Malik, J. (2018). End-to-End Recovery of Human Shape and Pose. In *Conference on Computer Vision and Pattern Recognition*. (page 23)

[119] Kazemi, V. and Sullivan, J. (2014). One Millisecond Face Alignment with an Ensemble of Regression Trees. In *Conference on Computer Vision and Pattern Recognition*. (page 20)

[120] Kehl, W., Manhardt, F., Tombari, F., Ilic, S., and Navab, N. (2017). SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. In *International Conference on Computer Vision*. (page 119, 136, 160, 163, 164)

[121] Kehl, W., Milletari, F., Tombari, F., Ilic, S., and Navab, N. (2016). Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation. In *European Conference on Computer Vision*. (page 164)

[122] Keskin, C., Kıraç, F., Kara, Y. E., and Akarun, L. (2011). Real Time Hand Pose Estimation Using Depth Sensors. In *International Conference on Computer Vision*. (page 1, 2, 28, 69)

[123] Keskin, C., Kıraç, F., Kara, Y. E., and Akarun, L. (2012). Hand Pose Estimation and Hand Shape Classification Using Multi-Layered Randomized Decision Forests. In *European Conference on Computer Vision*. (page 28)

[124] Khamis, S., Taylor, J., Shotton, J., Keskin, C., Izadi, S., and Fitzgibbon, A. (2015). Learning an Efficient Model of Hand Shape Variation from Depth Images. In *Conference on Computer Vision and Pattern Recognition*. (page 25)

[125] Kingma, D. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *International Conference for Learning Representations*. (page 53, 81, 125, 140, 170)

[126] Kingma, D. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *International Conference for Learning Representations*. (page 136)

[127] Kraft, D. (1988). A Software Package for Sequential Quadratic Programming. Technical report, German Aerospace Center. (page 102, 105)

[128] Krejov, P., Gilbert, A., and Bowden, R. (2016). Guided Optimisation through Classification and Regression for Hand Pose Estimation. *Computer Vision and Image Understanding*, 155(2):124–138. (page 24, 26, 27, 61)

[129] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*. (page 47, 50, 57, 101)

[130] Krull, A., Brachmann, E., Michel, F., Yang, M. Y., Gumhold, S., and Rother, C. (2015). Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images. In *International Conference on Computer Vision*. (page 30, 160)

[131] Kuettel, D., Guillaumin, M., and Ferrari, V. (2012). Segmentation Propagation in ImageNet. In *European Conference on Computer Vision*. (page 102)

[132] Kulkarni, T. D., Whitney, W., Kohli, P., and Tenenbaum, J. B. (2015). Deep Convolutional Inverse Graphics Network. In *Advances in Neural Information Processing Systems*. (page 71)

[133] Kulkarni, T. D., Yildirim, I., Kohli, P., Freiwald, W. A., and Tenenbaum, J. B. (2014). Deep Generative Vision as Approximate Bayesian Computation. In *Advances in Neural Information Processing Systems*. (page 71)

[134] Kuznetsova, A., Leal-Taixe, L., and Rosenhahn, B. (2013). Real-Time Sign Language Recognition Using a Consumer Depth Camera. In *International Conference on Computer Vision*. (page 28)

[135] Lai, H., Xiao, S., Pan, Y., Cui, Z., Feng, J., Xu, C., Yin, J., and Yan, S. (2018). Deep Recurrent Regression for Facial Landmark Detection. *Transactions on Circuits and Systems for Video Technology*, 28(5):1144–1157. (page 20)

[136] Lai, K., Bo, L., Ren, X., and Fox, D. (2011). A Scalable Tree-Based Approach for Joint Object and Pose Recognition. In *American Association for Artificial Intelligence Conference*. (page 141)

[137] Leap Motion (2018). Leap Motion. `https://www.leapmotion.com/`. Accessed: 2018-08-08. (page 10)

[138] Lee, S. (2014). Time-of-Flight Depth Camera Motion Blur Detection and Deblurring. *IEEE Signal Processing Letters*, 21(6):663–666. (page 9)

[139] Lepetit, V. and Berger, M. (2000). A Semi-Automatic Method for Resolving Occlusions in Augmented Reality. In *Conference on Computer Vision and Pattern Recognition*. (page 99)

[140] Levenberg, K. (1944). A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics*, 2(2):164–168. (page 105, 106)

[141] Li, C., Zeeshan Zia, M., Tran, Q.-H., Yu, X., Hager, G. D., and Chandraker, M. (2017). Deep Supervision With Shape Concepts for Occlusion-Aware 3D Object Parsing. In *Conference on Computer Vision and Pattern Recognition*. (page 118)

[142] Li, P., Ling, H., Li, X., and Liao, C. (2015). 3D Hand Pose Estimation Using Randomized Decision Forest with Segmentation Index Points. In *International Conference on Computer Vision*. (page 29)

[143] Li, S. and Lee, D. (2017). 2017 HANDS Challenge: Team HCR. (page 65)

[144] Li, Y., Wang, G., Ji, X., Xiang, Y., and Fox, D. (2018). DeepIM: Deep Iterative Matching for 6D Pose Estimation. In *European Conference on Computer Vision*. (page 91)

[145] Liang, H., Yuan, J., Thalmann, D., and Zhang, Z. (2013). Model-based Hand Pose Estimation via Spatial-temporal Hand Parsing and 3D Fingertip Localization. *The Visual Computer*, 29(6):837–848. (page 25, 26, 27)

[146] Liebelt, J., Schmid, C., and Schertler, K. (2008). Independent Object Class Detection Using 3D Feature Maps. In *Conference on Computer Vision and Pattern Recognition*. (page 118)

[147] Lien, J., Gillian, N., Karagozler, M. E., Amihood, P., Schwesig, C., Olson, E., Raja, H., and Poupyrev, I. (2016). Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar. *ACM Transactions on Graphics*, 35(4):142:1–142:19. (page 9)

[148] Lin, C.-H. and Lucey, S. (2017). Inverse Compositional Spatial Transformer Networks. In *Conference on Computer Vision and Pattern Recognition*. (page 151)

[149] Lin, J., Wu, Y., and Huang, T. S. (2000). Modeling the Constraints of Human Hand Motion. In *Workshop on Human Motion*. (page 2)

[150] Liu, C., Yuen, J., and Torralba, A. (2011). SIFT Flow: Dense Correspondence Across Scenes and Its Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):978–994. (page 104)

[151] Liu, S., Liang, X., Liu, L., Shen, X., Yang, J., Xu, C., Lin, L., Cao, X., and Yan, S. (2015). Matching-CNN Meets KNN: Quasi-Parametric Human Parsing. In *Conference on Computer Vision and Pattern Recognition*. (page 77)

[152] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In *European Conference on Computer Vision*. (page 164)

[153] Long, M., Cao, Y., Wang, J., and Jordan, M. I. (2015). Learning Transferable Features with Deep Adaption Networks. In *International Conference on Machine Learning*. (page 120)

[154] Long, M., Zhu, H., Wang, J., and Jordan, M. I. (2016). Unsupervised Domain Adaptation with Residual Transfer Networks. In *Advances in Neural Information Processing Systems*. (page 120)

[155] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110. (page 20, 30, 163)

[156] Madadi, M., Escalera, S., Baro, X., and Gonzalez, J. (2017a). End-To-End Global to Local CNN Learning for Hand Pose Recovery in Depth Data. In *arXiv Preprint*. (page 29, 59, 65)

[157] Madadi, M., Escalera, S., Carruesco, A., Andujar, C., Baro, X., and Gonzalez, J. (2017b). Occlusion Aware Hand Pose Recovery from Sequences of Depth Images. In *Automated Face and Gesture Recognition*. (page 30)

[158] Manhardt, F., Kehl, W., Navab, N., and Tombari, F. (2018). Deep Model-Based 6D Pose Refinement in RGB. In *European Conference on Computer Vision*. (page 91)

[159] Márquez-Neila, P., Salzmann, M., and Fua, P. (2017). Imposing Hard Constraints on Deep Networks: Promises and Limitations. In *Conference on Computer Vision and Pattern Recognition*. (page 184)

[160] Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. (2001). Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In *International Conference on Artificial Neural Networks*. (page 100)

[161] Massa, F., Russell, B., and Aubry, M. (2016). Deep Exemplar 2D-3D Detection by Adapting from Real to Rendered Views. In *Conference on Computer Vision and Pattern Recognition*. (page 119, 121, 130)

[162] McLain, T. M. (2010). The Use of Factor Analysis in the Development of Hand Sizes for Glove Design. *Industrial and Management Systems Engineering – Dissertations and Student Research*, 4. (page 7, 182)

[163] Mehta, D., Rhodin, H., Casas, D., Fua, P., Sotnychenko, O., Xu, W., and Theobalt, C. (2017a). Monocular 3D Human Pose Estimation in the Wild Using Improved CNN Supervision. *International Conference on 3D Vision*. (page 22)

[164] Mehta, D., Sridhar, S., Sotnychenko, O., Rhodin, H., Shafiei, M., Seidel, H.-P., Xu, W., Casas, D., and Theobalt, C. (2017b). VNect: Real-time 3D Human Pose Estimation with a Single RGB Camera. *ACM Transactions on Graphics*, 36(4):1–14. (page 23)

[165] Melax, S., Keselman, L., and Orsten, S. (2013). Dynamics Based 3D Skeletal Hand Tracking. In *Graphics Interface Conference*. (page 1, 2, 10, 25, 26, 27, 89, 98)

[166] Microsoft (2018). Windows Mixed Reality. `https://docs.microsoft.com/en-us/windows/mixed-reality/`. Accessed: 2018-08-08. (page 10)

[167] Mitash, C., Boularias, A., and Bekris, K. E. (2018). Improving 6D Pose Estimation of Objects in Clutter via Physics-aware Monte Carlo Tree Search. In *International Conference on Robotics and Automation*. (page 164)

[168] Moeslund, T. B. and Granum, E. (2001). A Survey of Computer Vision-Based Human Motion Capture. *Computer Vision and Image Understanding*, 81(3):231–268. (page 21)

[169] Molchanov, P., Kautz, J., and Honari, S. (2017). 2017 HANDS Challenge: Team NVIDIA Research and UMontreal. (page 64, 65)

[170] Moon, G., Chang, J. Y., and Lee, K. M. (2018). V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation from a Single Depth Map. In *Conference on Computer Vision and Pattern Recognition*. (page 7, 8, 29, 64, 65)

[171] Muandet, K., Balduzzi, D., and Schölkopf, B. (2013). Domain Generalization via Invariant Feature Representation. In *International Conference on Machine Learning*. (page 118, 120, 135, 136)

[172] Müller, F., Bernard, F., Sotnychenko, O., Mehta, D., Sridhar, S., Casas, D., and Theobalt, C. (2018). GANerated Hands for Real-Time 3D Hand Tracking from Monocular RGB. In *Conference on Computer Vision and Pattern Recognition*. (page 2, 30, 37, 63, 64, 133, 136, 137, 140, 141, 142)

[173] Müller, F., Mehta, D., Sotnychenko, O., Sridhar, S., Casas, D., and Theobalt, C. (2017). Real-Time Hand Tracking Under Occlusion From an Egocentric RGB-D Sensor. In *International Conference on Computer Vision*. (page 10, 28, 30, 74, 125, 139, 142, 183)

[174] Nair, P. and Cavallaro, A. (2009). 3-D Face Detection, Landmark Localization, and Registration Using a Point Distribution Model. *Transactions on Multimedia*, 11(4):611–623. (page 19)

[175] Nair, V., Susskind, J., and Hinton, G. E. (2008). Analysis-By-Synthesis by Learning to Invert Generative Black Boxes. In *International Conference on Artificial Neural Networks*. (page 12, 71)

[176] Neidle, C., Thangali, A., and Sclaroff, S. (2012). Challenges in Development of the American Sign Language Lexicon Video Dataset (ASLLVD) Corpus. In *Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon*. (page 3, 4)

[177] Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. (1978). An Analysis of Approximations for Maximizing Submodular Set Functions - I. *Mathematical Programming*, 14(1):265–294. (page 100)

[178] Neverova, N., Wolf, C., Nebout, F., and Taylor, G. (2017). Hand Pose Estimation through Semi-Supervised and Weakly-Supervised Learning. *Computer Vision and Image Understanding*, 164:56–67. (page 10, 28, 29, 59, 84, 127, 128)

[179] Newell, A., Yang, K., and Deng, J. (2016). Stacked Hourglass Networks for Human Pose Estimation. In *European Conference on Computer Vision*. (page 2)

[180] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer. (page 105)

[181] Oberweger, M. and Lepetit, V. (2017). DeepPrior++: Improving Fast and Accurate 3D Hand Pose Estimation. In *International Conference on Computer Vision Workshops*. (page 11, 74, 125, 134, 139, 142)

[182] Oberweger, M., Riegler, G., Wohlhart, P., and Lepetit, V. (2016). Efficiently Creating 3D Training Data for Fine Hand Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 11)

[183] Oberweger, M., Wohlhart, P., and Lepetit, V. (2015a). Hands Deep in Deep Learning for Hand Pose Estimation. In *Computer Vision Winter Workshop*. (page 11, 36, 60, 69, 74, 83, 96, 127)

[184] Oberweger, M., Wohlhart, P., and Lepetit, V. (2015b). Training a Feedback Loop for Hand Pose Estimation. In *International Conference on Computer Vision*. (page 11, 91, 95)

[185] Oculus (2018). Oculus Rift. `https://www.oculus.com/rift/`. Accessed: 2018-08-08. (page 10)

[186] Oikonomidis, I., Kyriazis, N., and Argyros, A. A. (2011a). Efficient Model-Based 3D Tracking of Hand Articulations Using Kinect. In *British Machine Vision Conference*. (page 10, 26, 69, 85, 89)

[187] Oikonomidis, I., Kyriazis, N., and Argyros, A. A. (2011b). Full DOF Tracking of a Hand Interacting with an Object by Modeling Occlusions and Physical Constraints. In *International Conference on Computer Vision*. (page 1, 10, 23, 24, 25, 26, 27, 30, 69, 148)

[188] Oikonomidis, I., Kyriazis, N., and Argyros, A. A. (2012). Tracking the Articulated Motion of Two Strongly Interacting Hands. In *Conference on Computer Vision and Pattern Recognition*. (page 30, 31)

[189] Oikonomidis, I., Lourakis, M. I. A., and Argyros, A. A. (2014). Evolutionary Quasi-Random Search for Hand Articulations Tracking. In *Conference on Computer Vision and Pattern Recognition*. (page 24, 26)

[190] Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2014). Learning and Transferring Mid-Level Image Representations Using Convolutional Neural Networks. In *Conference on Computer Vision and Pattern Recognition*. (page 120)

[191] Osherov, E. and Lindenbaum, M. (2017). Increasing CNN Robustness to Occlusions by Reducing Filter Support. In *International Conference on Computer Vision*. (page 164)

[192] Pan, S., Tsang, I., Kwok, J., and Yang, Q. (2009). Domain Adaptation via Transfer Component Analysis. In *International Joint Conference on Artificial Intelligence*. (page 118, 120, 135, 136)

[193] Panteleris, P., Kyriazis, N., and Argyros, A. A. (2015). 3D Tracking of Human Hands in Interaction with Unknown Objects. In *British Machine Vision Conference*. (page 31)

[194] Panteleris, P., Oikonomidis, I., and Argyros, A. A. (2018). Using a Single RGB Frame for Real Time 3D Hand Pose Estimation in the Wild. In *IEEE Winter Conference on Applications of Computer Vision*. (page 30)

[195] Pavlakos, G., Zhou, X., Chan, A., Derpanis, K. G., and Daniilidis, K. (2018). 6-DoF Object Pose from Semantic Keypoints. In *International Conference on Intelligent Robots and Systems*. (page 160, 163, 164)

[196] Pavlakos, G., Zhou, X., Derpanis, K. G., and Daniilidis, K. (2017). Coarse-to-Fine Volumetric Prediction for Single-Image 3D Human Pose. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[197] Plänkers, R. and Fua, P. (2003). Articulated Soft Objects for Multi-View Shape and Motion Capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1182–1187. (page 69)

[198] Pons-Moll, G., Fleet, D. J., and Rosenhahn, B. (2014). Posebits for Monocular Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 101)

[199] Poppe, R. (2007). Vision-Based Human Motion Analysis: An Overview. *Computer Vision and Image Understanding*, 108(1):4–18. (page 21)

[200] PrimeSense (2015). Primesense 3D Sensors. (page 77)

[201] Qian, C., Sun, X., Wei, Y., Tang, X., and Sun, J. (2014). Realtime and Robust Hand Tracking from Depth. In *Conference on Computer Vision and Pattern Recognition*. (page 1, 10, 24, 25, 26, 27, 69, 81, 85, 89, 98, 110, 153, 154)

[202] Quach, K. G., Duong, C. N., Luu, K., and Bui, T. D. (2016). Depth-Based 3D Hand Pose Tracking. In *International Conference on Pattern Recognition*. (page 30)

[203] Rad, M. and Lepetit, V. (2017). BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects Without Using Depth. In *International Conference on Computer Vision*. (page 11, 30, 91, 134, 136, 159, 160, 163, 164, 165, 166, 170, 171, 174, 175)

[204] Rad, M., Oberweger, M., and Lepetit, V. (2018a). Domain Transfer for 3D Pose Estimation from Color Images without Manual Annotations. In *Asian Conference on Computer Vision*. (page 134)

[205] Rad, M., Oberweger, M., and Lepetit, V. (2018b). Feature Mapping for Learning Fast and Accurate 3D Pose Inference from Synthetic Images. In *Conference on Computer Vision and Pattern Recognition*. (page 117, 134, 136, 137, 139)

[206] Ramanan, D. (2007). Learning to Parse Images of Articulated Bodies. In *Advances in Neural Information Processing Systems*. (page 22)

[207] Rautaray, S. S. and Agrawal, A. (2015). Vision based Hand Gesture Recognition for Human Computer Interaction: A Survey. *Artificial Intelligence Review*, 43(1):1–54. (page 2)

[208] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Conference on Computer Vision and Pattern Recognition*. (page 164)

[209] Rehg, J. M. and Kanade, T. (1994). DigitEyes: Vision-Based Hand Tracking for Human-Computer Interaction. In *IEEE Workshop on Motion of Non-rigid and Articulated Objects*. (page 1, 23)

[210] Ren, S., Cao, X., Wei, Y., and Sun, J. (2014). Face Alignment at 3000 FPS via Regressing Local Binary Features. In *Conference on Computer Vision and Pattern Recognition*. (page 20)

[211] Riegler, G., Ferstl, D., Rüther, M., and Bischof, H. (2014). Hough Networks for Head Pose Estimation and Facial Feature Localization. In *British Machine Vision Conference*. (page 165)

[212] Riegler, G., Ferstl, D., Rüther, M., and Bischof, H. (2015). A Framework for Articulated Hand Pose Estimation and Evaluation. In *Scandinavian Conference on Image Analysis*. (page 68, 96, 98, 106)

[213] Riegler, G., Ulusoy, A. O., and Geiger, A. (2017). OctNet: Learning Deep 3D Representations at High Resolution. In *Conference on Computer Vision and Pattern Recognition*. (page 29, 64)

[214] Rogez, G., Khademi, M., Supancic, J. S., Montiel, J. M. M., and Ramanan, D. (2014). 3D Hand Pose Detection in Egocentric RGB-D Images. In *European Conference on Computer Vision*. (page 28, 97, 98, 112)

[215] Rogez, G., Supancic, J. S., and Ramanan, D. (2015). Understanding Everyday Hands in Action from RGB-D Images. In *International Conference on Computer Vision*. (page 30, 96, 98)

[216] Romero, J., Kjellström, H., and Kragic, D. (2010). Hands in Action: Real-time 3D Reconstruction of Hands in Interaction with Objects. In *International Conference on Robotics and Automation*. (page 30)

[217] Rosales, R. and Sclaroff, S. (2006). Combining Generative and Discriminative Models in a Framework for Articulated Pose Estimation. *International Journal of Computer Vision*, 67(3):251–276. (page 27, 185)

[218] Rozantsev, A., Lepetit, V., and Fua, P. (2015). On Rendering Synthetic Images for Training an Object Detector. *Computer Vision and Image Understanding*, 137:24–37. (page 120)

[219] Rozantsev, A., Salzmann, M., and Fua, P. (2017). Beyond Sharing Weights for Deep Domain Adaptation. In *Conference on Computer Vision and Pattern Recognition*. (page 118, 120, 126, 127, 135, 136)

[220] Sapp, B. and Taskar, B. (2013). MODEC: Multimodal Decomposable Models for Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[221] Sauer, P., Cootes, T., and Taylor, C. (2011). Accurate Regression Procedures for Active Appearance Models. In *British Machine Vision Conference*. (page 20)

[222] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In *International Conference on Artificial Neural Networks*. (page 77)

[223] Schmidhuber, J. (2014). Deep Learning in Neural Networks: An Overview. Technical report, IDSIA. (page 45)

[224] Schmidt, T., Newcombe, R., and Fox, D. (2015). DART: Dense Articulated Real-time Tracking with Consumer Depth Cameras. *Autonomous Robots*, 39(3):239–258. (page 10, 24, 26, 27)

[225] Schröder, M., Maycock, J., Ritter, H., and Botsch, M. (2014). Real-Time Hand Tracking using Synergistic Inverse Kinematics. In *International Conference on Robotics and Automation*. (page 25, 26, 27)

[226] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2014). OverFeat: Integrated Recognition, Localization and Detection Using Convolutional Networks. In *International Conference for Learning Representations*. (page 47, 48)

[227] Sharp, T., Keskin, C., Robertson, D., Taylor, J., Shotton, J., Kim, D., Rhemann, C., Leichter, I., Vinnikov, A., Wei, Y., Freedman, D., Kohli, P., Krupka, E., Fitzgibbon, A., and Izadi, S. (2015). Accurate, Robust, and Flexible Real-Time Hand Tracking. In *ACM Conference on Human Factors in Computing Systems*. (page 10, 25, 26, 27, 69, 85, 185)

[228] Shimada, N., Shirai, Y., Kuno, Y., and Miura, J. (1998). Hand Gesture Estimation and Model Refinement Using Monocular Camera-ambiguity Limitation by Inequality Constraints. In *Automated Face and Gesture Recognition*. (page 27)

[229] Shotton, J., Girshick, R., Fitzgibbon, A., Sharp, T., Cook, M., Finocchio, M., Moore, R., Kohli, P., Criminisi, A., Kipman, A., and Blake, A. (2011). Efficient Human Pose Estimation from Single Depth Images. In *Conference on Computer Vision and Pattern Recognition*. (page 1, 2, 5, 21, 28)

[230] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., and Webb, R. (2016). Learning from Simulated and Unsupervised Images through Adversarial Training. In *Conference on Computer Vision and Pattern Recognition*. (page 119, 121, 135)

[231] Sigal, L. and Black, M. J. (2006). Measure Locally, Reason Globally: Occlusion-sensitive Articulated Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[232] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference for Learning Representations*. (page 51, 119)

[233] Sinha, A., Choi, C., and Ramani, K. (2016). Deephand: Robust Hand Pose Estimation by Completing a Matrix Imputed with Deep Features. In *Conference on Computer Vision and Pattern Recognition*. (page 29)

[234] Sminchisescu, C. and Triggs, B. (2003). Kinematic Jump Processes for Monocular 3D Human Tracking. In *Conference on Computer Vision and Pattern Recognition*. (page 101)

[235] Soltanpour, S., Boufama, B., and Wu, Q. J. (2017). A Survey of Local Feature Methods for 3D Face Recognition. *Pattern Recognition*, 72:391–406. (page 20)

[236] Song, X., Jiang, S., and Herranz, L. (2017). Combining Models from Multiple Sources for RGB-D Scene Recognition. In *International Joint Conference on Artificial Intelligence*. (page 136, 137)

[237] Spurr, A., Song, J., Park, S., and Hilliges, O. (2018). Cross-modal Deep Variational Hand Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 30, 37, 133, 136, 140, 141, 142)

[238] Sridhar, S., Feit, A. M., Theobalt, C., and Oulasvirta, A. (2015a). Investigating the Dexterity of Multi-Finger Input for Mid-Air Text Entry. In *ACM Conference on Human Factors in Computing Systems*. (page 185)

[239] Sridhar, S., Müller, F., Oulasvirta, A., and Theobalt, C. (2015b). Fast and Robust Hand Tracking using Detection-guided Optimization. In *Conference on Computer Vision and Pattern Recognition*. (page 7, 25, 70)

[240] Sridhar, S., Müller, F., Zollhoefer, M., Casas, D., Oulasvirta, A., and Theobalt, C. (2016). Real-time Joint Tracking of a Hand Manipulating an Object from RGB-D Input. In *European Conference on Computer Vision*. (page 10, 27, 31, 37, 38, 40, 147, 148, 154, 155, 156, 157, 183)

[241] Sridhar, S., Oulasvirta, A., and Theobalt, C. (2013). Interactive Markerless Articulated Hand Motion Tracking Using RGB and Depth Data. In *International Conference on Computer Vision*. (page 24, 25, 26, 27, 69, 81, 89, 96, 98, 183)

[242] Sridhar, S., Rhodin, H., Seidel, H.-P., Oulasvirta, A., and Theobalt, C. (2014). Real-time Hand Tracking Using a Sum of Anisotropic Gaussians Model. In *International Conference on 3D Vision*. (page 25)

[243] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958. (page 51, 52)

[244] Stark, M., Goesele, M., and Schiele, B. (2010). Back to the Future: Learning Shape Models from 3D CAD Data. In *British Machine Vision Conference*. (page 118)

[245] Sugimura, S. and Hoshino, K. (2017). Wearable Hand Pose Estimation for Remote Control of a Robot on the Moon. *Journal of Robotics and Mechatronics*, 29(5):829–837. (page 3, 4)

[246] Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *International Conference on Computer Vision*. (page 119)

[247] Sun, X., Wei, Y., Liang, S., Tang, X., and Sun, J. (2015). Cascaded Hand Pose Regression. In *Conference on Computer Vision and Pattern Recognition*. (page 7, 8, 29, 33, 35, 36, 50, 57, 61, 62, 63, 95, 96, 98, 106, 110, 111, 112, 113, 153, 154)

[248] Sun, Y., Wang, X., and Tang, X. (2013). Deep Convolutional Network Cascade for Facial Point Detection. In *Conference on Computer Vision and Pattern Recognition*. (page 36, 46, 49)

[249] Supancic, J. S., Rogez, G., Yang, Y., Shotton, J., and Ramanan, D. (2015). Depth-Based Hand Pose Estimation: Data, Methods, and Challenges. In *International Conference on Computer Vision*. (page 28, 36, 39, 46, 59, 60, 96, 112, 114, 115)

[250] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going Deeper with Convolutions. In *Conference on Computer Vision and Pattern Recognition*. (page 51)

[251] Tagliasacchi, A., Schröder, M., Tkach, A., Bouaziz, S., Botsch, M., and Pauly, M. (2015). Robust Articulated-ICP for Real-time Hand Tracking. *Computer Graphics Forum*, 34(5):101–114. (page 23, 24, 26, 30, 59, 61, 70)

[252] Tan, D. J., Cashman, T., Taylor, J., Fitzgibbon, A., Tarlow, D., Khamis, S., Izadi, S., and Shotton, J. (2016). Fits Like a Glove: Rapid and Reliable Hand Shape Personalization. In *Conference on Computer Vision and Pattern Recognition*. (page 7, 25, 59)

[253] Tang, D. (2015). *3D Hand Pose Regression with Variants of Decision Forests*. PhD thesis, Imperial College London. (page 39)

[254] Tang, D., Chang, H. J., Tejani, A., and Kim, T.-K. (2014a). Latent Regression Forest: Structured Estimation of 3D Articulated Hand Posture. In *Conference on Computer Vision and Pattern Recognition*. (page 7, 8, 10, 28, 33, 35, 36, 46, 47, 52, 53, 57, 60, 61, 62, 69, 74, 81, 82, 95, 96, 98, 153)

[255] Tang, D., Taylor, J., Kohli, P., Keskin, C., Kim, T.-K., and Shotton, J. (2015). Opening the Black Box: Hierarchical Sampling Optimization for Estimating Human Hand Pose. In *International Conference on Computer Vision*. (page 25, 26, 60)

[256] Tang, D., Yu, T., and Kim, T.-K. (2013). Real-Time Articulated Hand Pose Estimation Using Semi-Supervised Transductive Regression Forests. In *International Conference on Computer Vision*. (page 1, 2, 28, 69)

[257] Tang, Y., Srivastava, N., and Salakhutdinov, R. (2014b). Learning Generative Models with Visual Attention. In *Advances in Neural Information Processing Systems*. (page 71)

[258] Taylor, C. J. (2000). Reconstruction of Articulated Objects from Point Correspondences in a Single Uncalibrated Image. In *Conference on Computer Vision and Pattern Recognition*. (page 101)

[259] Taylor, J., Bordeaux, L., Cashman, T., Corish, B., Keskin, C., Sharp, T., Soto, E., Sweeney, D., Valentin, J., Luff, B., Topalian, A., Wood, E., Khamis, S., Kohli, P., Izadi, S., Banks, R., Fitzgibbon, A., and Shotton, J. (2016). Efficient and Precise Interactive Hand Tracking through Joint, Continuous Optimization of Pose and Correspondences. *ACM Transactions on Graphics*, 35(4):143:1–143:12. (page 4, 5, 23, 24, 25, 27, 59, 61, 69, 70)

[260] Taylor, J., Shotton, J., Sharp, T., and Fitzgibbon, A. (2012). The Vitruvian Manifold: Inferring Dense Correspondences for One-Shot Human Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 21, 40, 53, 127)

[261] Taylor, J., Stebbing, R., Ramakrishna, V., Keskin, C., Shotton, J., Izadi, S., Hertzmann, A., and Fitzgibbon, A. (2014). User-Specific Hand Modeling from Monocular Depth Sequences. In *Conference on Computer Vision and Pattern Recognition*. (page 25)

[262] Teimourikia, M., Saidinejad, H., Comai, S., and Salice, F. (2014). Personalized Hand Pose and Gesture Recognition System for the Elderly. In *International Conference on Universal Access in Human-Computer Interaction*. (page 182)

[263] Tekin, B., Katircioglu, I., Salzmann, M., Lepetit, V., and Fua, P. (2016). Structured Prediction of 3D Human Pose with Deep Neural Networks. In *British Machine Vision Conference*. (page 22, 68)

[264] Tekin, B., Sinha, S. N., and Fua, P. (2018). Real-Time Seamless Single Shot 6D Object Pose Prediction. In *Conference on Computer Vision and Pattern Recognition*. (page 134, 159, 160, 163, 164, 171, 174)

[265] Tkach, A., Pauly, M., and Tagliasacchi, A. (2016). Sphere-meshes for Real-time Hand Modeling and Tracking. *ACM Transactions on Graphics*, 35(6):1–11. (page 24, 25, 27)

[266] Tkach, A., Tagliasacchi, A., Remelli, E., Pauly, M., and Fitzgibbon, A. (2017). Online Generative Model Personalization for Hand Tracking. *ACM Transactions on Graphics*, 36(6):243:1–243:11. (page 7, 25, 69, 70)

[267] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In *International Conference on Intelligent Robots and Systems*. (page 136, 137)

[268] Tolani, D., Goswami, A., and Badler, N. I. (2000). Real-time Inverse Kinematics Techniques for Anthropomorphic Limbs. *Graphical Models*, 62(5):353–388. (page 3)

[269] Tomasi, C., Petrov, S., and Sastry, A. (2003). 3D Tracking = Classification + Interpolation. In *International Conference on Computer Vision*. (page 8)

[270] Tompson, J., Jain, A., LeCun, Y., and Bregler, C. (2014a). Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation. In *Advances in Neural Information Processing Systems*. (page 22, 47)

[271] Tompson, J., Stein, M., LeCun, Y., and Perlin, K. (2014b). Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks. *ACM Transactions on Graphics*, 33(5):1–10. (page 2, 7, 8, 10, 11, 28, 33, 34, 35, 36, 46, 50, 53, 54, 55, 57, 59, 60, 61, 65, 66, 67, 69, 81, 82, 83, 84, 90, 92, 95, 96, 98, 112, 117, 118, 119, 126, 127, 128, 129, 130, 131, 132, 140)

[272] Toshev, A. and Szegedy, C. (2014). DeepPose: Human Pose Estimation via Deep Neural Networks. In *Conference on Computer Vision and Pattern Recognition*. (page 22, 46, 47, 49)

[273] Tresadern, P., Sauer, P., and Cootes, T. (2010). Additive Update Predictors in Active Appearance Models. In *British Machine Vision Conference*. (page 20)

[274] Tzeng, E., Hoffman, J., Saenko, K., and Darrell, T. (2017). Adversarial Discriminative Domain Adaptation. In *Conference on Computer Vision and Pattern Recognition*. (page 121, 126)

[275] Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., and Darell, T. (2014). Deep Domain Confusion: Maximizing for Domain Invariance. In *arXiv Preprint*. (page 120, 126, 127)

[276] Tzionas, D., Ballan, L., Srikantha, A., Aponte, P., Pollefeys, M., and Gall, J. (2016). Capturing Hands in Action using Discriminative Salient Points and Physics Simulation. In *International Journal of Computer Vision*. (page 31, 147, 148)

[277] Tzionas, D. and Gall, J. (2013). A Comparison of Directional Distances for Hand Pose Estimation. In *German Conference on Pattern Recognition*. (page 26, 96, 98)

[278] Tzionas, D. and Gall, J. (2015). 3D Object Reconstruction from Hand-Object Interactions. In *International Conference on Computer Vision*. (page 31)

[279] Tzionas, D., Srikantha, A., Aponte, P., and Gall, J. (2014). Capturing Hand Motion with an RGB-D Sensor, Fusing a Generative Model with Salient Points. In *German Conference on Pattern Recognition*. (page 10, 25, 26, 27, 69)

[280] van der Maaten, L. and Hinton, G. (2008). Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605. (page 34, 101)

[281] Wachs, J. P., Kölsch, M., Stern, H., and Edan, Y. (2011). Vision-based Hand-gesture Applications. *Communications of the ACM*, 54(2):60–71. (page 4, 5)

[282] Wagner, D., Langlotz, T., and Schmalstieg, D. (2008). Robust and Unobtrusive Marker Tracking on Mobile Phones. In *International Symposium on Mixed and Augmented Reality*. (page 30)

[283] Wan, C., Probst, T., Van Gool, L., and Yao, A. (2017). Crossing Nets: Dual Generative Models with a Shared Latent Space for Hand Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 29, 36, 59, 61, 62, 65, 72, 84, 127, 128)

[284] Wan, C., Probst, T., Van Gool, L., and Yao, A. (2018). Dense 3D Regression for Hand Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 28)

[285] Wan, C., Yao, A., and Van Gool, L. (2016). Hand Pose Estimation from Local Surface Normals. In *European Conference on Computer Vision*. (page 29, 60, 61)

[286] Wang, J., Xie, Q., Zhang, Z., Zhu, J., Xie, L., and Yuille, A. L. (2017). Detecting Semantic Parts on Partially Occluded Objects. In *British Machine Vision Conference*. (page 163, 164)

[287] Wang, N., Gao, X., Tao, D., Yang, H., and Lie, X. (2018). Facial Feature Point Detection: A Comprehensive Survey. *Neurocomputing*, 275:50–65. (page 20)

[288] Wang, R. Y., Paris, S., and Popovic, J. (2011). 6D Hands: Markerless Hand Tracking for Computer Aided Design. In *User Insterface Software and Technology Symposium*. (page 30)

[289] Wang, R. Y. and Popovic, J. (2009). Real-Time Hand-Tracking with a Color Glove. *ACM Transactions on Graphics*, 28(3):63:1–63:8. (page 27)

[290] Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional Pose Machines. In *Conference on Computer Vision and Pattern Recognition*. (page 2, 21, 159, 165, 166, 174, 175)

[291] Wei, X. and Chai, J. (2010). VideoMocap: Modeling Physically Realistic Human Motion from Monocular Video Sequences. *ACM Transactions on Graphics*, 29(4):42:1–42:10. (page 99)

[292] Wetzler, A., Slossberg, R., and Kimmel, R. (2015). Rule of Thumb: Deep Derotation for Improved Fingertip Detection. In *British Machine Vision Conference*. (page 98)

[293] Wohlhart, P. and Lepetit, V. (2015). Learning Descriptors for Object Recognition and 3D Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*. (page 30)

[294] Wu, Y. and Ji, Q. (2015). Robust Facial Landmark Detection under Significant Head Poses and Occlusion. In *International Conference on Computer Vision*. (page 20)

[295] Wu, Y. and Ji, Q. (2018). Facial Landmark Detection: A Literature Survey. *International Journal of Computer Vision*. (page 5)

[296] Wu, Y., Lin, J. Y., and Huang, T. S. (2001). Capturing Natural Hand Articulation. In *International Conference on Computer Vision*. (page 47, 48, 97)

[297] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2018). PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *Robotics: Science and Systems Conference*. (page 37, 38, 41, 134, 151, 160, 162, 163, 164, 170, 171, 174, 175, 176, 177, 178, 179, 180)

[298] Xiong, X. and la Torre, F. D. (2013). Supervised Descent Method and Its Applications to Face Alignment. In *Conference on Computer Vision and Pattern Recognition*. (page 20)

[299] Xu, C. and Cheng, L. (2013). Efficient Hand Pose Estimation from a Single Depth Image. In *International Conference on Computer Vision*. (page 1, 23, 25, 26, 27, 81, 89, 96, 98)

[300] Xu, C., Govindarajan, L. N., Zhang, Y., and Cheng, L. (2017). Lie-X: Depth Image Based Articulated Object Pose Estimation, Tracking, and Action Recognition on Lie Groups. *International Journal of Computer Vision*, 123(3):454–478. (page 10, 28, 59, 84, 127, 128)

[301] Xu, W., Chatterjee, A., Zollhoefer, M., Rhodin, H., Fua, P., Seidel, H.-P., and Theobalt, C. (2018). Mo2Cap2: Real-time Mobile 3D Motion Capture with a Cap-mounted Fisheye Camera. In *arXiv Preprint*. (page 22)

[302] Yang, F., Akiyama, K., and Wu, Y. (2017). 2017 HANDS Challenge: Team NAIST RV. (page 65)

[303] Yang, H. and Zhang, J. (2016). Hand Pose Regression via a Classification-Guided Approach. In *Asian Conference on Computer Vision*. (page 29, 62)

[304] Yang, Y. and Ramanan, D. (2011). Articulated Pose Estimation with Flexible Mixtures-of-Parts. In *Conference on Computer Vision and Pattern Recognition*. (page 22)

[305] Yang, Y. and Ramanan, D. (2013). Articulated Human Detection with Flexible Mixtures of Parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2878–2890. (page 40)

[306] Yasin, H., Iqbal, U., Krüger, B., Weber, A., and Gall, J. (2016). A Dual-Source Approach for 3D Pose Estimation from a Single Image. In *Conference on Computer Vision and Pattern Recognition*. (page 98)

[307] Ye, Q., Yuan, S., and Kim, T. (2016). Spatial Attention Deep Net with Partial PSO for Hierarchical Hybrid Hand Pose Estimation. In *European Conference on Computer Vision*. (page 29)

[308] Yi, Z., Zhang, H., Tan, P., and Gong, M. (2017). DualGAN: Unsupervised Dual Learning for Image-to-Image Translation. In *International Conference on Computer Vision*. (page 121)

[309] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How Transferable are Features in Deep Neural Networks? In *Advances in Neural Information Processing Systems*. (page 120)

[310] Yuan, S., Garcia-Hernando, G., Stenger, B., Moon, G., Chang, J. Y., Lee, K. M., Molchanov, P., Kautz, J., Honari, S., Ge, L., Yuan, J., Chen, X., Wang, G., Yang, F., Akiyama, K., Wu, Y., Wan, Q., Madadi, M., Escalera, S., Li, S., Lee, D., Oikonomidis, I., Argyros, A. A., and Kim, T.-K. (2018). Depth-Based 3D Hand Pose Estimation: From Current Achievements to Future Goals. In *Conference on Computer Vision and Pattern Recognition*. (page 64)

[311] Yuan, S., Ye, Q., Stenger, B., Jain, S., and Kim, T.-K. (2017). BigHand2.2M Benchmark: Hand Pose Dataset and State of the Art Analysis. In *Conference on Computer Vision and Pattern Recognition*. (page 33, 34, 35, 37, 57, 64, 65, 134, 182)

[312] Zakharov, S., Planche, B., Wu, Z., Hutter, A., Kosch, H., and Ilic, S. (2018). Keep it Unreal: Bridging the Realism Gap for 2.5D Recognition with Geometry Priors Only. In *arXiv Preprint*. (page 136, 137)

[313] Zamir, A. R., Wu, T.-L., Sun, L., Shen, W., Malik, J., and Savarese, S. (2017). Feedback Networks. In *Conference on Computer Vision and Pattern Recognition*. (page 72)

[314] Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive Deconvolutional Networks for Mid and High Level Feature Learning. In *International Conference on Computer Vision*. (page 76, 100, 168)

[315] Zhang, H. and Cao, Q. (2017). Combined Holistic and Local Patches for Recovering 6D Object Pose. In *International Conference on Computer Vision Workshops*. (page 29, 59, 164)

[316] Zhang, J., Jiao, J., Chen, M., Qu, L., Xu, X., and Yang, Q. (2017). A Hand Pose Tracking Benchmark from Stereo Matching. In *International Conference on Image Processing*. (page 30, 33, 35, 37, 57, 63, 64, 135, 140, 141, 142, 143)

[317] Zhang, Y., Wang, S., and Ji, G. (2015). A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. *Mathematical Problems in Engineering*, 2015. (page 26)

[318] Zhang, Z., Luo, P., Loy, C. C., and Tang, X. (2016). Learning Deep Representation for Face Alignment with Auxiliary Attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(5):918–930. (page 21)

[319] Zhao, R., Wang, Y., and Martinez, A. (2017). A Simple, Fast and Highly-Accurate Algorithm to Recover 3D Shape from 2D Landmarks on a Single Image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. (page 141)

[320] Zhou, X. and Jacobson, A. (2016). Thingi10K: A Dataset of 10,000 3D-Printing Models. In *arXiv Preprint*. (page 119)

[321] Zhou, X., Wan, Q., Zhang, W., Xue, X., and Wei, Y. (2016). Model-Based Deep Hand Pose Estimation. In *International Joint Conference on Artificial Intelligence*. (page 10, 28, 29, 59, 61, 84, 127, 128)

[322] Zhou, Y., Lu, J., Du, K., Lin, X., Sun, Y., and Ma, X. (2018). HBE: Hand Branch Ensemble Network for Real-time 3D Hand Pose Estimation. In *European Conference on Computer Vision*. (page 68)

[323] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *International Conference on Computer Vision*. (page 119, 121, 135, 136)

[324] Zhu, X., Lei, Z., Liu, X., Shi, H., and Li, S. Z. (2016). Face Alignment Across Large Poses: A 3D Solution. In *Conference on Computer Vision and Pattern Recognition*. (page 20)

[325] Zimmermann, C. and Brox, T. (2017). Learning to Estimate 3D Hand Pose from Single RGB Images. In *International Conference on Computer Vision*. (page 2, 10, 28, 30, 37, 63, 64, 68, 133, 140, 141, 142)

[326] Zuffi, S., Romero, J., Schmid, C., and Black, M. J. (2013). Estimating Human Pose with Flowing Puppets. In *International Conference on Computer Vision*. (page 23)