

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

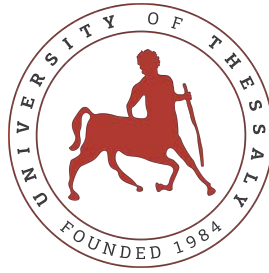
Articulated 3D Hand from a single RGB image

Diploma Thesis

Christos Pantazopoulos

Supervisor: Gerasimos Potamianos

February 2025



UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Articulated 3D Hand from a single RGB image

Diploma Thesis

Christos Pantazopoulos

Supervisor: Gerasimos Potamianos

February 2025



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αρθρωτό 3Δ χέρι από απλή εικόνα χρώματος

Διπλωματική Εργασία

Χρήστος Πανταζόπουλος

Επιβλέπων: Γεράσιμος Ποταμιάνος

Φεβρουάριος 2025

Approved by the Examination Committee:

Supervisor **Gerasimos Potamianos**

Associate Professor, Dept. of Electrical and Computer Engineering,
University of Thessaly

Member **Elias N. Houstis**

Emeritus Professor, Dept. of Electrical and Computer Engineering,
University of Thessaly

Member **Dimitrios Katsaros**

Associate Professor, Dept. of Electrical and Computer Engineering,
University of Thessaly

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Gerasimos Potamianos, for his guidance, insightful feedback, and support throughout this thesis. His expertise and advice have been invaluable in shaping my research and improving my work.

A special thanks to Dr. Spyridon Thermos, whose collaboration and extensive help played a crucial role in this thesis. His expertise, patience, and constant willingness to assist made a significant impact, and I truly appreciate his support from the very beginning.

I am also grateful to Dr. Elias Houstis for his encouragement throughout my academic years and for serving on my thesis committee, as well as to Dr. Dimitrios Katsaros for kindly accepting to be part of the committee.

A heartfelt thank you to my family—my parents, Giannis and Georgia, and my sister, Eleni—for their unconditional love, encouragement, and belief in me.

Finally, I am deeply thankful to my friends from university for their constant support, motivation, and for making this journey both memorable and fulfilling.

This thesis would not have been possible without all of you.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Christos Pantazopoulos

Diploma Thesis

Articulated 3D Hand from a single RGB image

Christos Pantazopoulos

Abstract

Reconstructing a 3D articulated hand from a single RGB image is a continuously investigated problem in computer vision, with applications in Augmented Reality (AR), Virtual Reality (VR), Human-Computer Interaction (HCI), and robotics. Apart from the absence of depth information, the possible occlusions and the articulation complexity pose additional challenges. This thesis investigates the efficiency of different optimization methods in fitting a parametric and continuously differentiable 3D hand model to a simple 2D image. In particular, we use a state-of-the-art (SotA) 2D hand pose estimation model to detect the human hand in the input image and extract 2D keypoints that represent the locations of the hand joints. The localized keypoints are then used in the fitting process as constraints. This process incorporates 3 steps, namely the 2D keypoints extraction, the shape initialization of the hand and its alignment with the detected hand, and the iterative optimization for aligning the parametric hand model's joints with the detected keypoints. The latter includes a novel fingertip loss that improves the fitting overall accuracy.

Experiments on the EgoDexter and Dexter+Object datasets show that our approach performs competitively with the SotA and that it is robust on real-world images without camera parameters. Failure analysis highlights sensitivity to keypoints accuracy, especially in occlusion scenarios. Future work includes extending the method to multiple hands and improving efficiency for real-time applications.

Keywords:

Articulated 3D hand, parametric hand model, MANO, keypoint detection, MediaPipe, optimization, computer vision

Διπλωματική Εργασία

Αρθρωτό 3Δ χέρι από απλή εικόνα χρώματος

Χρήστος Πανταζόπουλος

Περίληψη

Η ανακατασκευή ενός τρισδιάστατου (3Δ) αρθρωτού χεριού από μία μόνο εικόνα RGB αποτελεί μια δύσκολη πρόκληση στην όραση υπολογιστών, με εφαρμογές στην επαυξημένη πραγματικότητα (AR), την εικονική πραγματικότητα (VR), την αλληλεπίδραση ανθρώπου-υπολογιστή (HCI) και τη ρομποτική. Εκτός από την απουσία πληροφορίας βάθους, οι πιθανές αποκρύψεις και η πολυπλοκότητα των αρθρώσεων του χεριού εισάγουν επιπλέον προκλήσεις στο πρόβλημα. Η παρούσα διπλωματική εργασία εξετάζει την αποδοτικότητα διαφορετικών μεθόδων βελτιστοποίησης για την προσαρμογή ενός παραμετρικού και συνεχώς διαφορίσιμου 3Δ μοντέλου χεριού σε μια απλή 2Δ εικόνα.

Συγκεκριμένα, χρησιμοποιούμε ένα σύγχρονο (SotA) μοντέλο εκτίμησης 2Δ θέσης του χεριού για τον εντοπισμό του ανθρώπινου χεριού στην είσοδο της εικόνας και την εξαγωγή 2Δ σημείων-κλειδιών που αντιπροσωπεύουν τις θέσεις των αρθρώσεων του χεριού και χρησιμοποιούνται στη διαδικασία προσαρμογής ως περιορισμοί. Η διαδικασία αυτή περιλαμβάνει τρία στάδια: την εξαγωγή των 2Δ σημείων-κλειδιών, την αρχικοποίηση του σχήματος του χεριού και την ευθυγράμμιση του με το ανιχνευμένο χέρι, καθώς και την επαναληπτική βελτιστοποίηση για την ευθυγράμμιση των αρθρώσεων του παραμετρικού μοντέλου χεριός με τα ανιχνευμένα σημεία-κλειδιά. Στο τελευταίο στάδιο ενσωματώνουμε μια νέα συνάρτηση απωλειών για τις άκρες των δακτύλων, που βελτιώνει την ακρίβεια της προσαρμογής.

Τα πειράματα στα σύνολα δεδομένων EgoDexter και Dexter+Object δείχνουν ότι η προσέγγισή μας αποδίδει ανταγωνιστικά σε σχέση με τη SotA και ότι είναι ανθεκτική σε πραγματικές εικόνες χωρίς παραμέτρους κάμερας. Η ανάλυση αποτυχιών αναδεικνύει την ευαισθησία στην ακρίβεια των σημείων-κλειδιών, ειδικά σε σενάρια απόκρυψης. Μελλοντικές επεκτάσεις περιλαμβάνουν την προσαρμογή της μεθόδου για την αναγνώριση πολλαπλών χεριών και τη βελτίωση της αποδοτικότητας για εφαρμογές σε πραγματικό χρόνο.

Λέξεις-κλειδιά:

Αρθρωτό 3Δ χέρι, παραμετρικό μοντέλο χεριού, MANO, ανίχνευση βασικών σημείων, MediaPipe, βελτιστοποίηση, υπολογιστική όραση

Table of contents

Acknowledgements	ix
Abstract	xii
Περίληψη	xiii
Table of contents	xv
List of figures	xix
List of tables	xxiii
Abbreviations	xxv
1 Introduction	1
1.1 Problem Challenges and Thesis Approach	1
1.2 Thesis Contribution	2
1.3 Thesis Organization	3
2 Related Work	5
2.1 2D hand pose estimation	6
2.2 3D Hand Pose Estimation	9
2.3 Using Parametric Models	9
3 Hand Parametric Models	15
3.1 MANO	15
3.2 Step-by-step MANO Hand Visualization	18
3.3 manotorch: MANO PyTorch	18

3.4	Other Hand Parametric Models	19
4	Image-Based Hand Pose Estimation	21
4.1	Hand Keypoint Detection Models	21
4.1.1	MediaPipe	21
4.1.2	OpenPose	21
4.1.3	MMPose	22
4.2	Models Comparison	22
4.3	Our Approach	23
4.4	The ReRun Viewer	24
5	Optimization - Theoretical Background	25
5.1	Optimization Methods	25
5.1.1	BFGS Algorithm	26
5.1.2	Limited-memory BFGS (L-BFGS)	27
5.2	Loss Functions	27
5.2.1	Mean Squared Error	27
5.2.2	Geman-McClure	28
5.2.3	Huber	29
5.2.4	Anatomical Priors	29
5.3	Implementation Details	30
6	Proposed Method	33
6.1	Overview	33
6.2	Problems Encountered during Development	34
6.3	Theoretical Background on Rigid Transformation	36
6.3.1	Computing the Optimal Rigid Transformation	37
6.4	Description of the Optimization Pipeline	38
6.4.1	Extracting Key Features	38
6.4.2	Scaling and Aligning. Rigid Transformation as initialization	39
6.4.3	Optimization	39
6.4.4	Loss functions	40
6.4.5	Reversing Transformation and Visualization	41
6.4.6	Using Images in the Wild	41

7	Evaluation and Results	43
7.1	Quantitative Evaluation	43
7.1.1	Metrics	43
7.1.2	Data	44
7.1.3	Preprocessing	48
7.1.4	Ablation Study	50
7.1.5	Comparison with SotA Methods	55
7.2	Qualitative Evaluation	56
8	Conclusion	61
8.1	Summary	61
8.2	Future Work	62
	Bibliography	63
	Appendix	
		71
A.1	Custom python loss functions	71
A.2	Rigid transformation	72

List of figures

2.1	Left: skeleton visualization. Right: hand keypoints. (Figure source: [1]). . .	7
2.2	A typical 2D hand pose estimator. (Figure source: [1]).	7
2.3	Multiview bootstrapping. (a) a multiview system provides views of the hand where keypoint detection is easy, which are used to triangulate (b) the 3D position of the keypoints. Difficult views with (c) failed detections can be (d) annotated using the reprojected 3D keypoints, and used to retrain (e) an improved detector that now works on difficult views. (Figure source: [2]). .	8
2.4	Examples of 3D hand pose estimation and visualization by [3] on different datasets without using parametric models. (Figure source: [3]).	10
2.5	The Boukhayma <i>et al.</i> [4] pipeline takes as input a hand image and optionally 2D joint heatmaps from an independent CNN. The encoder generates the shape, pose, and view parameters. The hand parameters are fed to the hand model that generates a triangulated 3D mesh and its underlying 3D skeleton. The latter are re-projected into the image domain using a weak perspective camera model controlled by the view parameters. This network is trained end-to-end with a combination of weak 2D and full 3D joint supervision. The hand and view parameters are not supervised. (Figure source: [4]). . .	11
2.6	Example of a synthetic hand image dataset creation. From left to right a 3D hand mesh is rendered with texture and proper lighting, a background image is chosen, and the rendered hand and the random background are blended to create a new, unseen hand-related image. (Figure source: [5]).	12

2.7	The framework overview of [6]. From left to right: (a) a cropped image of a hand is passed through a convolutional encoder to extract key features which are sent to an iterative regression module to infer a set of parameters, such as shape and joint rotations. (b) These parameters are then used by the hand model to reconstruct the 3D hand mesh and pose. (c) The generated 3D joint positions are then projected to a 2D plane using the camera intrinsics. The depicted network can be trained end-to-end using both 2D and 3D keypoints supervision. (Figure source: [6]).	13
3.1	Data collection of [7]. Their system captures a hand with the wrist resting on a hand stand, and performing different articulated motions, resulting in a high quality, textured scan of the hand seen on the right. (Figure source: [7]).	16
3.2	Hand capture protocol of [7]: 31 subjects performed most of the 51 shown poses. (Figure source: [7]).	16
3.3	Examples of the captured poses for a single subject in [7]. (Figure source: [7]).	16
3.4	Examples of the captured “flat-hand” pose for various subjects in [7]. (Figure source: [7]).	17
4.1	21 hand keypoints in MediaPipe. (Figure source: [8]).	22
4.2	Example of using MediaPipe hand landmark model. (Figure source: [8]). . .	22
5.1	Kinematical structure and joint notations. (Figure source: [9]).	31
6.1	Our proposed pipeline	33
6.2	Initialization and optimization pipeline	38
7.1	3D PCK for EgoDexter dataset.	53
7.2	3D PCK for Dexter+Object dataset.	54
7.3	Results on an image with a simple hand pose from the EgoDexter dataset. The method performs well in this scenario, accurately capturing the keypoints.	57
7.4	Results on an image from the EgoDexter dataset featuring a challenging hand pose with occlusion due to object interaction. Despite the occlusion, the method provides reasonable predictions.	57

7.5	Results on an image from the Dexter+Object dataset using a simple MSE loss function. The predicted hand pose shows small inaccuracies, particularly in fingertip locations.	58
7.6	Results on the same Dexter+Object image as Figure 7.5, but using a weighted MSE loss function with fingertip emphasis. The hand pose estimation is significantly improved, demonstrating the advantage of this loss function in specific cases.	58
7.7	Example from the EgoDexter dataset where our method fails due to severe occlusion. The dataset does not provide ground truth keypoints in this particular case. Meanwhile, MediaPipe also fails to detect the hand.	59
7.8	Hand pose estimation on an image of the famous Mona Lisa painting. This demonstrates that our method generalizes to standard RGB images and can work on images in the wild, independent of camera parameters.	59

List of tables

7.1	A collection of datasets comprising optical annotated data for hand pose estimation. “#J” is the number of joints annotated, and “ Obj int ” states whether there is object interaction.	45
7.2	Additional information of the datasets.	46
7.3	The description of what combination of optimizer and loss function each experiment uses	51
7.4	Ablation Study: End-Point Error (mm) (↓) and AUC of PCK (↑) results of EgoDexter and Dexter+Object.	52
7.5	AUC of PCK (↑) comparison with state-of-the-art methods. We use “*” to note the methods that work in real-time that are more challenging.	55

Abbreviations

AR	Augmented Reality
AUC	Area Under the Curve
BFGS	Broyden-Fletcher-Goldfarb-Shanno
CNN	Convolutional Neural Network
DIP	Distal InterPhalangeal joint
DNN	Deep Neural Network
DoF	Degrees of Freedom
EPE	End-Point Error
GAN	Generative Adversarial Network
GM	Geman-McClure
IP	InterPhalangeal joint
L-BFGS	Limited memory Broyden-Fletcher-Goldfarb-Shanno
MAE	Mean Absolute Error
MANO	hand Model with Articulated and Non-rigid defOrmations
MCP	MetaCarpophalangeal joint
MP	MediaPipe
MSE	Mean Squared Error
PCA	Principal Component Analysis
PCK	Percentage of Correct Keypoints
PIP	Proximal InterPhalangeal joint
RMSE	Root Mean Squared Error
SVD	Singular Value Decomposition
TM	TrapezioMetacarpal joint
VR	Virtual Reality

Chapter 1

Introduction

Reconstructing an articulated 3D hand from a single RGB image is a challenging problem in computer vision with a wide range of applications, including augmented and virtual reality (AR/VR), human-computer interaction (HCI), robotics, and the metaverse. The ability to estimate 3D hand poses from 2D images enables natural interaction in digital environments and is crucial for remote control of robotic systems. However, achieving accurate 3D reconstruction without depth information remains a difficult task due to several inherent challenges.

1.1 Problem Challenges and Thesis Approach

Estimating a 3D hand pose from a single RGB image is difficult because a 2D image does not provide explicit depth information. Unlike structured depth sensors, which capture spatial geometry, color images represent a two-dimensional perspective of the 3D world, making it challenging to infer the correct 3D articulation of the hand. Additionally, hands exhibit high flexibility with 21 degrees of freedom (DoF), and they often experience occlusions—either self-occlusion, where fingers block each other, or occlusion caused by external objects during hand-object interactions. These factors introduce ambiguities, making direct 3D hand pose estimation from images a challenging task.

To address this, we experiment with different optimization methods to fit a parametric and continuously differentiable 3D hand model to a simple 2D image based on estimated keypoints. The key idea is to transform the problem from inferring absolute depth values to an optimization task that aligns a predefined 3D model to observed 2D features. The parametric model we use, the “hand Model with Articulated and Non-rigid defOrmations” (MANO),

provides a compact way to represent any possible hand shape and pose using a set of predefined parameters. Specifically, it uses 10 shape parameters to describe variations in hand size and proportions, and 45 pose parameters to define joint rotations and hand articulation.

Our approach follows these steps:

1. Extract 2D keypoints from an input image using a SotA hand detection and pose estimation model.
2. Use a parametric and continuously differentiable hand model to generate a 3D hand with adjustable shape and pose parameters.
3. Optimize the alignment between the detected 2D keypoints and the 3D hand model by adjusting the model's parameters.

By fitting the 3D hand model to the extracted 2D keypoints, we effectively transform the problem from estimating 3D joint rotations to finding the correct joint positions in space. This formulation avoids the need for large-scale training datasets and provides a flexible, optimization-based approach for 3D hand reconstruction.

This solution has practical applications in robotics (remote control of robotic arms), AR/VR (natural hand interaction in virtual environments), HCI (gesture-based control), and the metaverse, where realistic hand tracking enhances immersion in digital spaces.

1.2 Thesis Contribution

The contributions of this thesis can be summarized as follows:

1. We develop an optimization-based approach to fit an articulated 3D hand from a single RGB image without requiring camera parameters or extensive training on large datasets.
2. We integrate the MediaPipe model [10] for 2D hand keypoint detection with the MANO parametric hand model [7], creating a pipeline that takes as optimization constraints the 2D keypoints and fits 3D hands through an optimization process.
3. We use anatomical priors as a loss function, as well as a weighted fingertips loss function to improve the accuracy of our method, addressing a limitation in 3D hand fitting methods.

4. We conduct extensive quantitative and qualitative evaluations on the EgoDexter [11] and Dexter+Object [12] datasets, demonstrating that our method achieves competitive performance compared to SotA learning-based approaches.
5. We show that our approach generalizes well to real-world images, including those taken in uncontrolled environments, proving that it can function without requiring known camera parameters.
6. We conduct an extensive literature review on 2D and 3D hand pose estimation methods. Based on this study, we identify key challenges in recovering 3D hands from a single RGB image and design an accurate solution.

1.3 Thesis Organization

The thesis begins with a review of existing research in Chapter 2 about 2D and 3D hand pose estimation, covering both 3D hand pose estimation with and without the use of parametric models. This discussion highlights the challenges in recovering an articulated 3D hand from a single RGB image and sets the foundation for our approach.

Next, in Chapter 3 we delve into parametric hand models, with a particular focus on the MANO model. We describe its structure, visualization process, and related models, emphasizing their role in 3D hand reconstruction. Since our method relies on 2D keypoint detection, we also explore different hand keypoint detection frameworks in Chapter 4, comparing MediaPipe, OpenPose, and MMPose before justifying our choice of MediaPipe for this work. Additionally, we discuss visualization techniques using the ReRun Viewer.

To provide the necessary theoretical background, in Chapter 5 we examine key optimization methods and loss functions used in our approach. This includes an exploration of BFGS and L-BFGS algorithms, as well as different loss functions such as Mean Squared Error, Geman-McClure, and Huber loss. We also introduce anatomical constraints that help improve the accuracy of our optimization.

Building on this foundation, in Chapter 6 we present our proposed method, detailing the full pipeline from keypoint extraction to 3D hand reconstruction. The rigid transformation step, choice of loss functions, and challenges encountered during implementation are also discussed.

The evaluation Chapter 7 then presents both quantitative and qualitative results, including an ablation study on different components of our approach and comparisons with SotA methods. Additionally, we introduce the datasets used in our study, providing an overview of datasets used in the literature, and explaining our selection of the EgoDexter and Dexter+Object datasets for evaluation.

The thesis concludes with a summary of our findings in Chapter 8 and a discussion on possible future research directions, such as extending the method to multi-hand scenarios and improving computational efficiency for real-time applications. The final sections include the bibliography and the appendix, which provides additional implementation details.

Chapter 2

Related Work

Hand pose estimation refers to predicting the position and orientation of the hand and fingers in relation to a set coordinate system using either RGB images, volumetric data from depth cameras, or a combination of both. It is vital for VR and AR systems, gesture-based HCI and 3D motion capture. Additionally, it enhances user experience in VR and is crucial to various applications including gesture recognition, action recognition, and sign language recognition.

Despite significant progress, hand pose estimation is still a challenging task due to the complex nature of the physiology of hands and hardware limitations. Some of the most important challenges are:

- **Complex Articulation:** We need a lot of parameters to model the complexity of the human hand, as it has more than 20 DoF.
- **Occlusions:** Occlusions in general are a challenge in modeling objects. This is worse when it comes to human hands, as during the performance of a gesture, fingers of the same hand may be strongly occluded by each other, other body parts, or objects.
- **Low resolution:** The hand is usually not the focus of attention in images, and we need to crop them to only contain the hand. Consequently, the result is an image with lower resolution. Additionally, there are other limitations concerning hardware such as cameras and sensors.
- **Annotated data:** The creation of annotated 3D data is a costly and time-consuming task. In order to accurately capture 3D data, an expensive marker-based motion capture system or a massive multi-view camera setting is required.

- **Rapid hand and finger movements:** Usually, hand motion consists of very fast movements, sudden stops, and joint rotations that are not met in other body parts. This results in blurry frames or uncorrelated consecutive frames due to the inability of conventional cameras to capture more than 60 frames per second.

Neural networks have revolutionized computer vision tasks like image classification, object detection, and human pose estimation, yielding SotA results. Recent end-to-end Deep Neural Networks (DNNs) extract more meaningful features than traditional techniques, enhancing performance, especially with complex data like RGB images. This advancement has accelerated hand pose estimation, addressing challenges like hand localization through DNN utilization.

This study focuses on implementations that exploit solely the color information of the human hand.

2.1 2D hand pose estimation

In 2D hand pose estimation, to estimate the pose of the hand we need to estimate the location of its keypoints. The human hand has 21 keypoints [1]:

$$1 \text{ wrist} + 5 \text{ fingers} \times (3 \text{ finger joints} + 1 \text{ fingertip}) = 21$$

Their order is important, because we need to know the location of every unique feature in these 21 keypoints to retrieve the pose of the hand.

Hand pose is often visualized as a skeleton, with each finger skeleton shown in a different color. However, this skeleton is drawn just for visualization purposes by connecting the ordered keypoints with lines. In Figure 2.1 we see the keypoints on a hand and the visualization of a skeleton.

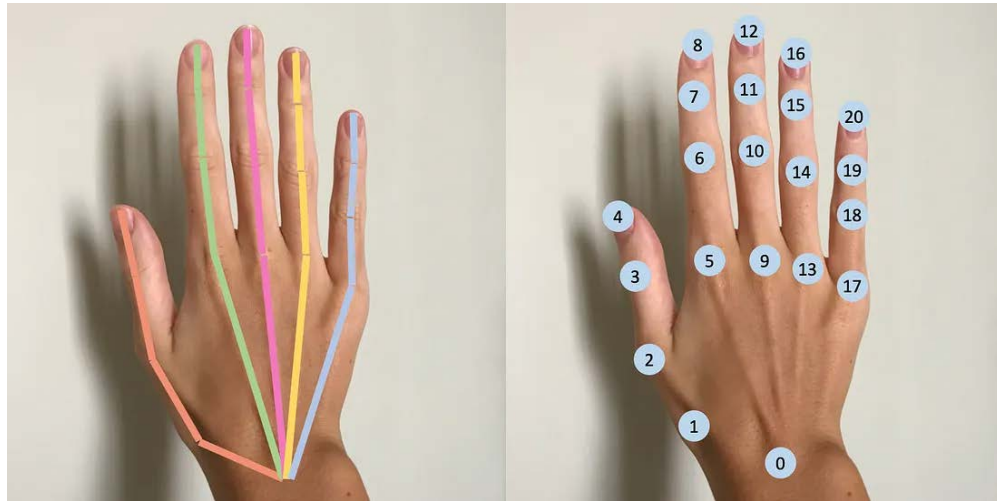


Figure 2.1: Left: skeleton visualization. Right: hand keypoints. (Figure source: [1]).

The input and output of a typical 2D hand pose estimator are:

- **Input:** hand image. For most hand pose estimators there is only one hand in the input image, and the image is cropped to exclude other body parts and background.
- **Output:** a list of (x, y) 2D keypoint locations.

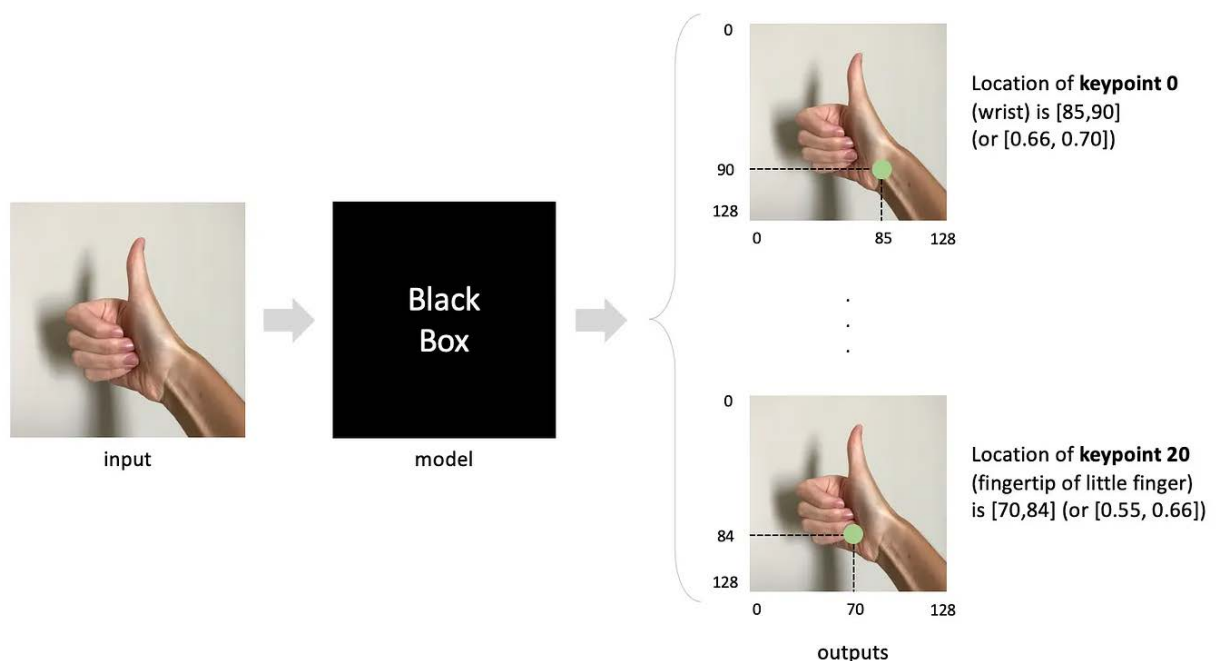


Figure 2.2: A typical 2D hand pose estimator. (Figure source: [1]).

It is clear that the right and the left hand have different features to estimate either when the view of the images is egocentric or third-person. However, in both views the right and the

left hands have mirror symmetry. For this reason, we usually train pose estimators to work for right hands of a third-person view. We then just mirror the image of the left hand so that it looks exactly like the right one to estimate its pose. Moreover, when annotated data are created to train these estimators, images of the left hand are also mirrored and get included to the training dataset.

There is a plethora of datasets with different characteristics that are listed later in Chapter 7. There are also many models that use depth information for a 2D hand pose estimator, but these are not included in this study.

In prior work on this topic, in [2] Simon *et al.* present an approach that uses a multi-camera system to train fine-grained detectors for keypoints that are prone to occlusion, such as the joints of a hand. This procedure is called multiview bootstrapping: first, an initial keypoint detector is used to produce noisy labels in multiple views of the hand. The noisy detections are then triangulated in 3D using multiview geometry or marked as outliers. Finally, the re-projected triangulations are used as new labeled training data to improve the detector. This process is repeated, generating more labeled data in each iteration. The method is used to train a hand keypoint detector for single images. The resulting keypoint detector runs in real-time on RGB images and has accuracy comparable to methods that use depth sensors.

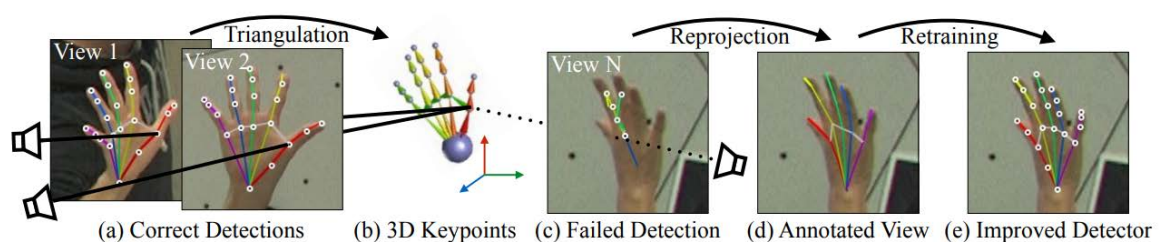


Figure 2.3: Multiview bootstrapping. (a) a multiview system provides views of the hand where keypoint detection is easy, which are used to triangulate (b) the 3D position of the keypoints. Difficult views with (c) failed detections can be (d) annotated using the reprojected 3D keypoints, and used to retrain (e) an improved detector that now works on difficult views. (Figure source: [2]).

2.2 3D Hand Pose Estimation

Estimating the 3D pose of a hand is an essential part of HCI. There have been numerous methods estimating 3D pose using depth or multi-view sensors. However, regressing pose from a single RGB image is challenging due to the fact that 3D pose requires some form of depth estimates, which are ambiguous given only an RGB image. This was later dealt with the use of parametric hand models [7] (see also Chapter 3) like we do in our proposed method. In this section we examine methods of 3D hand pose estimation without the use of parametric models.

Iqbal *et al.* [3] propose a new method for 3D hand pose estimation from a monocular image through a novel 2.5D pose representation. This new representation estimates pose up to a scaling factor, which can be estimated additionally if a prior of the hand size is given. They implicitly learn depth maps and heatmap distributions with a novel Convolutional Neural Network (CNN) architecture, achieving SotA estimation of 2D and 3D hand pose on several challenging datasets in the presence of severe occlusions. In Figure 2.4 we see examples of their results and visualizations.

Zimmermann and Brox [13] also present an approach that estimates 3D hand pose from RGB images. To handle the “missing” depth data they propose a deep network that learns a network-implicit 3D articulation prior. Together with detected keypoints in the images, this network yields good estimates of the 3D pose. Additionally, Zimmermann *et al.* [14] introduce a large-scale 3D hand pose dataset based on synthetic hand models for training the involved networks. There, they introduce the first large-scale, multi-view hand dataset that is accompanied by both 3D hand pose and shape annotations. For annotating this real-world dataset, an iterative, semi-automated ‘human-in-the-loop’ approach is proposed, which includes hand fitting optimization to infer both the 3D pose and shape for each sample.

2.3 Using Parametric Models

A stepping stone to the evolution of the 3D hand pose estimation from a single RGB image without the use of depth information or other cameras/sensors has been the introduction of hand parametric models like MANO [7]. This model enabled researchers to use it either for creating more realistic synthetic data or for implementing methods that predict MANO parameters to reconstruct the pose and shape of the hand in 3D.

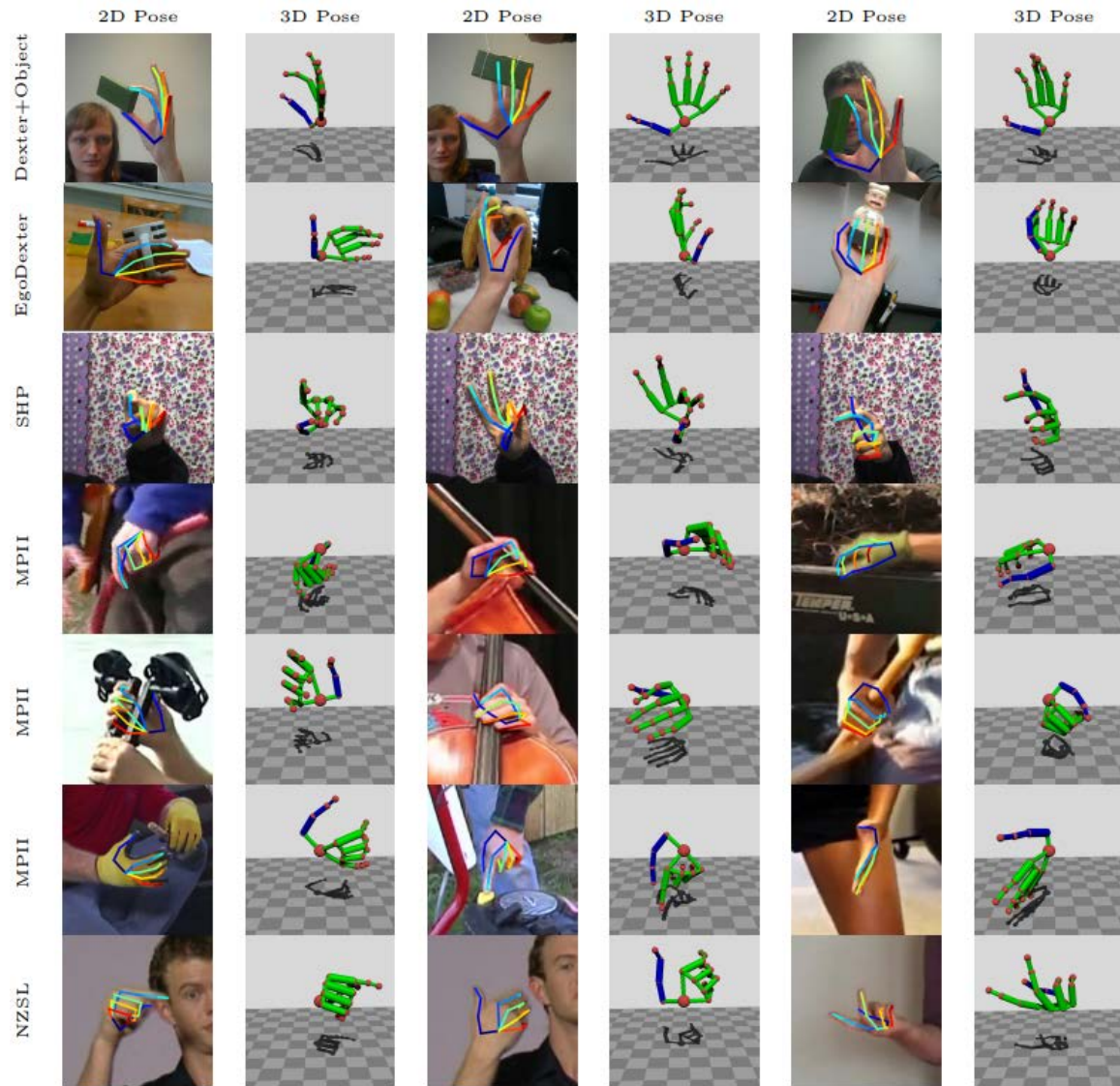


Figure 2.4: Examples of 3D hand pose estimation and visualization by [3] on different datasets without using parametric models. (Figure source: [3]).

Boukhayma *et al.* [4] present the first end-to-end deep learning based method that predicts both 3D hand shape and pose from RGB images in the wild. This network consists of the concatenation of a deep convolutional encoder and a fixed model-based decoder. Given an input image and, optionally 2D joint detections obtained from an independent CNN, the encoder predicts a set of hand and view parameters. The decoder has two components: a pre-computed articulated mesh deformation hand model that generates a 3D mesh from the hand parameters, and a re-projection module controlled by the view parameters that projects the generated hand into the image domain. It is also shown that using the shape and pose prior knowledge encoded in the hand model within a deep learning framework yields SotA

performance in 3D pose prediction from images on standard benchmarks, and produces geometrically valid and plausible 3D reconstructions. In Figure 2.5 we see the visualization of this pipeline.

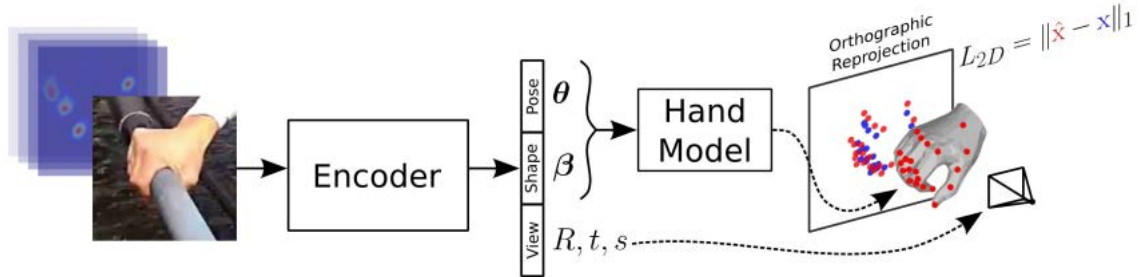


Figure 2.5: The Boukhayma *et al.* [4] pipeline takes as input a hand image and optionally 2D joint heatmaps from an independent CNN. The encoder generates the shape, pose, and view parameters. The hand parameters are fed to the hand model that generates a triangulated 3D mesh and its underlying 3D skeleton. The latter are re-projected into the image domain using a weak perspective camera model controlled by the view parameters. This network is trained end-to-end with a combination of weak 2D and full 3D joint supervision. The hand and view parameters are not supervised. (Figure source: [4]).

Panteleris *et al.* [15] present a method for the real-time estimation of the full 3D pose of one or more human hands using a single commodity RGB camera. More specifically, given an RGB image and the relevant camera calibration information, they employ a SotA detector to localize hands. Then, using a crop of a hand in the image, they run the pretrained network of OpenPose [2] for hands to estimate the 2D location of hand joints. Finally, non-linear least-squares minimization fits a 3D model of the hand to the estimated 2D joint positions, recovering the 3D hand pose.

Mueller *et al.* [16] address the problem of real-time 3D hand tracking based on a monocular RGB-only sequence. Their method combines a CNN with a kinematic 3D hand model. For training this CNN they generated a synthetic training dataset by using a neural network that translates synthetic images to “real” images, such that the so-generated images follow the same statistical distribution as real-world hand images.

Ge *et al.* [5] propose a Graph CNN based method to reconstruct a full 3D mesh of hand surface that contains richer information of both 3D hand shape and pose. To train networks with full supervision, they create a large-scale synthetic dataset containing both ground truth 3D meshes and 3D poses. When fine-tuning the networks on real-world datasets without 3D

ground truth, they propose a weakly-supervised approach by leveraging the depth map as a weak supervision in training.

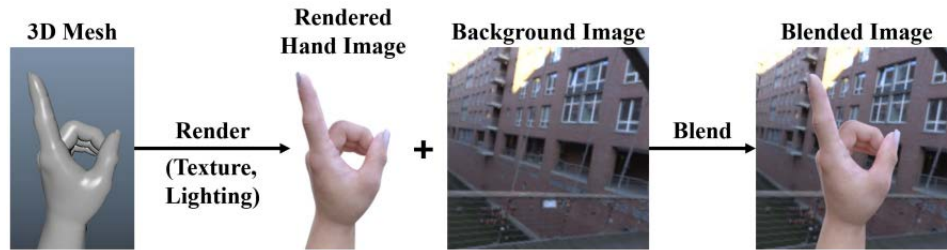


Figure 2.6: Example of a synthetic hand image dataset creation. From left to right a 3D hand mesh is rendered with texture and proper lighting, a background image is chosen, and the rendered hand and the random background are blended to create a new, unseen hand-related image. (Figure source: [5]).

Baek *et al.* [17] adopt the MANO parametric 3D hand model that represents hand meshes. To achieve the model fitting to RGB images they implement a hand mesh estimator by a neural network and a differentiable renderer, supervised by 2D segmentation masks and 3D skeletons. The method demonstrates good performance for estimating diverse hand shapes and improves pose estimation accuracies. Additionally, they iteratively refine the initial estimate using the gradients, in the spirit of iterative model fitting methods like the Iterative Closest Point (ICP) [18].

Kulon *et al.* [19] introduce a simple and effective network architecture for monocular 3D hand pose estimation consisting of an image encoder followed by a mesh convolutional decoder that is trained through a direct 3D hand mesh reconstruction loss. They train the network by gathering a large-scale dataset of hand action in YouTube videos and use it as a source of weak supervision.

Zhang *et al.* [20] present a Hand Mesh Recovery framework to tackle the problem of reconstructing the full 3D mesh of a human hand from a single RGB image. The mesh representation is achieved by parameterizing MANO, a generic 3D hand model with shape and relative 3D joint angles. The 3D joint locations are computed via linear interpolations between the vertices of the mesh, while the 2D joint locations are obtained with a projection of the 3D joints. To this end, a differentiable re-projection loss can be defined in terms of the derived representations and the ground-truth labels, thus making this framework end-to-end trainable.

In the most recent work [21] Drosakis and Argyros present a method for simultaneous 3D hand shape and pose estimation on a single RGB image frame. Specifically, their method fits the MANO 3D hand model to 2D hand keypoints. Fitting is achieved based on a novel 2D objective function that exploits anatomical joint limits, combined with shape regularization on the MANO hand model, jointly optimizing the 3D shape and pose of the hand in a single frame.

Lim *et al.* [6] present an approach for real-time estimation of 3D hand shape and pose from a single RGB image. To achieve real-time performance, an efficient CNN named MobileNetV3-Small is utilized to extract key features from an input image. The extracted features are then sent to an iterative 3D regression module to infer camera parameters, hand shapes, and joint angles for projecting and articulating a 3D hand model. By combining the deep neural network with the differentiable hand model, the network can be trained with supervision from 2D and 3D annotations in an end-to-end manner. Experiments on two publicly available datasets demonstrate that this approach matches the accuracy of most existing methods while running at over 110 Hz on a GPU or 75 Hz on a CPU. The framework overview can be seen at Figure 2.7.

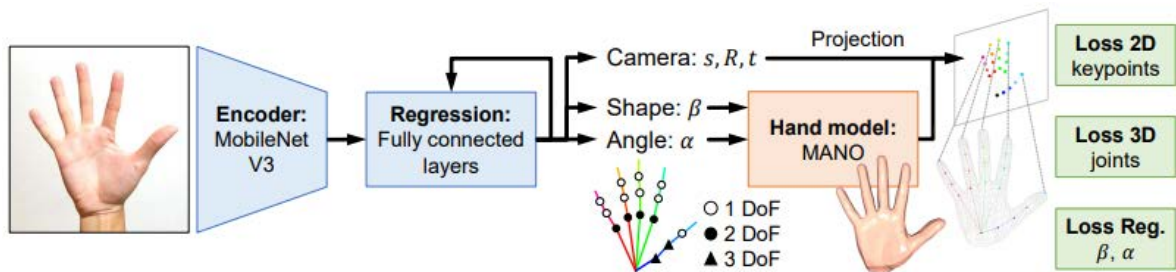


Figure 2.7: The framework overview of [6]. From left to right: (a) a cropped image of a hand is passed through a convolutional encoder to extract key features which are sent to an iterative regression module to infer a set of parameters, such as shape and joint rotations. (b) These parameters are then used by the hand model to reconstruct the 3D hand mesh and pose. (c) The generated 3D joint positions are then projected to a 2D plane using the camera intrinsics. The depicted network can be trained end-to-end using both 2D and 3D keypoints supervision. (Figure source: [6]).

Chapter 3

Hand Parametric Models

A 3D parametric model in computer vision describes objects or scenes using mathematical parameters, enabling accurate reconstruction and analysis of 3D structures from multiple viewpoints.

3.1 MANO

The MANO parametric model proposed by Romero *et al.* [7] is a parametric model commonly used in computer vision and graphics to encode the shape and pose variations of human hands. Principal Component Analysis (PCA) is utilized to capture the variations in hand shapes and poses from a set of training data, allowing for a compact representation of the hand's geometry and articulation. These parameters can then be used to generate realistic hand shapes and poses, enabling tasks such as hand tracking, animation, and synthesis in virtual environments.

MANO is learned from more than 1000 high-resolution 3D scans of hands of 31 subjects in a wide variety of hand poses. To further understand their process, in Figure 3.1, we see how the data collection was conducted. The system captures a hand with the wrist resting on a stand while performing various articulated motions. This setup ensures high-quality, textured scans that serve as the foundation for the MANO model. In Figure 3.2, we observe the hand capture protocol used to build the dataset. A total of 31 subjects performed most of the 51 predefined hand poses among others, providing a diverse set of examples that help in modeling realistic hand articulation. Figure 3.3 showcases examples of the captured poses for a single subject, while Figure 3.4 highlights variations of a specific pose, the "flat-hand"

pose, across multiple subjects. The top row presents the raw 3D scans, while the bottom row illustrates the corresponding mesh alignments.



Figure 3.1: Data collection of [7]. Their system captures a hand with the wrist resting on a hand stand, and performing different articulated motions, resulting in a high quality, textured scan of the hand seen on the right. (Figure source: [7]).

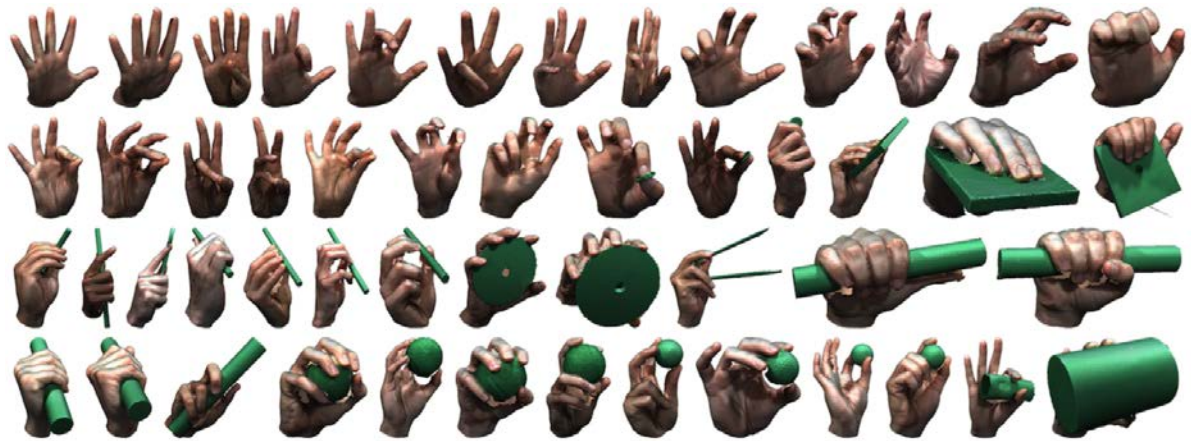


Figure 3.2: Hand capture protocol of [7]: 31 subjects performed most of the 51 shown poses. (Figure source: [7]).



Figure 3.3: Examples of the captured poses for a single subject in [7]. (Figure source: [7]).

The model is realistic, low-dimensional, and captures non-rigid shape changes with pose. It is compatible with standard graphics packages, and can fit any human hand. MANO provides a compact mapping from hand poses to pose blend shape corrections and a linear manifold of pose synergies.

The MANO hand model takes as input 45 rotation parameters θ and 10 shape parameters β to produce a 3D hand mesh. The MANO model is defined as a differentiable function as



Figure 3.4: Examples of the captured “flat-hand” pose for various subjects in [7]. (Figure source: [7]).

follows:

$$M(\beta, \theta) = W(T_p(\beta, \theta), J(\beta), \theta, G)$$

and

$$T_p(\beta, \theta) = \bar{T} + B_s(\beta) + B_P(\theta)$$

where $\beta \in \mathbb{R}^{10}$ and $\theta \in \mathbb{R}^{15 \times 3}$ are the hand shape and pose parameters, respectively, and W a linear blend skinning function applied to a template hand mesh T_p , which is obtained by deforming a mean hand mesh $\bar{T}$ with shape and pose blend shape functions B_s and B_P , respectively. Moreover, joints are denoted with J and blend weights with G . The resulting mesh $M \in \mathbb{R}^{N \times 3}$, where N denotes the number of vertices, is then posed in space with orientation $r \in \mathbb{R}^3$.

The MANO model provides a compact yet expressive representation of the human hand by utilizing both shape and pose parameters. The shape parameters, denoted as β , control the overall structure of the hand, such as the width of the fingers and palm. These parameters are derived through PCA on real hand scans, with only 10 parameters required to represent nearly all human hand shape variations.

On the other hand, the pose parameters, denoted as θ , define the rotations of the hand’s joints, allowing for various articulated poses. These rotations are represented in an axis-angle format for the 15 major joints of the hand, leading to a total of 45 values.

MANO data and models are available after signing up to the Project page¹ and has revolutionized 3D hand pose estimation. It is used for 3D reconstruction and the creation of synthetic datasets as we have already seen in Chapter 2.

¹<https://mano.is.tue.mpg.de/>

3.2 Step-by-step MANO Hand Visualization

To further understand and visualize some of the MANO hands, we can sign up and download the models. Then we can use one of the MANO frameworks in python and its functions to understand the parameters of the model and create a mesh of the hand based on our given parameters.

After installing the dependencies and importing the required libraries, we define a path to the MANO model directory that we earlier downloaded from the MANO Project page. Further, we need to specify other needed information such as the number of PCA coefficients or whether our model is based on right or left hands. Now that the model - denoted as Φ - has been loaded properly, we initialize the model by defining the following variables:

$$\Phi\{\beta, \theta, r\} \quad \text{with} \quad \beta \in \mathbb{R}^{10}, \theta \in \mathbb{R}^{45}, r \in \mathbb{R}^3 \quad (3.1)$$

which correspond to the beta, pose, global orientation and root translation parameters, respectively.

Now each model has different but similar functions available to obtain the joints, faces and vertices of the mesh in order to later visualize it using a visualization framework.

3.3 manotorch: MANO PyTorch

In the initial stages of development, we explored various software frameworks like [22], that implement the MANO model and provide basic functions for obtaining or setting its parameters. However, our final implementation is based on Manotorch [23].

Manotorch is a differentiable PyTorch layer that deterministically maps pose and shape parameters to hand joints and vertices using PyTorch. One key advantage of Manotorch is its built-in support for anatomical constraints through the Anatomy Loss function, which enforces biologically plausible hand configurations.

We selected Manotorch because integrating the Anatomy Loss function could be the solution to a key challenge encountered in the early stages of our implementation. To maximize flexibility, we decided to develop most of our pipeline from scratch within the PyTorch framework. This approach not only allowed us to incorporate Manotorch effectively but also provided access to a wider range of optimization techniques available in the PyTorch library.

3.4 Other Hand Parametric Models

There are also other hand parametric models available that are based on MANO and build other features on top of the MANO model hands, like texture or bones.

Potamias *et al.* [24] introduce Handy, a large-scale parametric model that improves upon MANO by incorporating both hand shape and appearance. While MANO was trained on a limited dataset of 31 adult subjects, Handy is built from over 1200 subjects, allowing it to better represent diversity in age, gender, and ethnicity. The Handy shape model follows a PCA-based structure similar to MANO but with higher fidelity, while the appearance model is learned using a powerful StyleGAN-v3 network, preserving fine details such as veins, wrinkles, and nail polish. To ensure compatibility, a version of the Handy model has been adapted to the MANO template, making it directly usable in existing applications that rely on MANO.

Qian *et al.* [25] present HTML, the first parametric texture model of human hands. This model spans several dimensions of hand appearance variability (e.g., related to gender, ethnicity, or age), only requires a commodity camera for data acquisition, and is based on MANO.

Li *et al.* [26] present PIANO, the first parametric bone model of human hands from MRI data. The PIANO model is biologically correct, simple to animate, and differentiable, achieving more anatomically precise modeling of the inner hand kinematic structure in a data-driven manner than traditional hand models based on outer surface only. Furthermore, the PIANO model can be applied in neural network layers to enable training with a fine-grained semantic loss, which opens up the new task of data-driven fine-grained hand bone anatomic and semantic understanding from MRI or even RGB images.

Chapter 4

Image-Based Hand Pose Estimation

4.1 Hand Keypoint Detection Models

The first step in our method is to detect and localize the 21 keypoints of the hand in the input image. This can be achieved using specialized frameworks that serve as keypoint encoders, such as MediaPipe [8], OpenPose [27], and MMPose [28]. Each of these frameworks offers different approaches and capabilities for hand pose estimation.

4.1.1 MediaPipe

The MediaPipe Hand Landmarker [10] lets you detect the landmarks of the hands in an image. You can use it to locate keypoints of hands and render visual effects on them. This task operates on image data with a machine learning model as static data or a continuous stream and outputs hand landmarks in image coordinates, hand landmarks in world coordinates, and handedness (left/right hand) of multiple detected hands.

The hand landmark model bundle detects the keypoint localization of 21 hand-knuckle coordinates within the detected hand regions. The model was trained on approximately 30k real-world images, as well as several rendered synthetic hand models imposed over various backgrounds.

4.1.2 OpenPose

OpenPose [27] constitutes the first real-time multi-person system to jointly detect human body, hand, facial, and foot keypoints (in total 135 keypoints) on single images. Among others

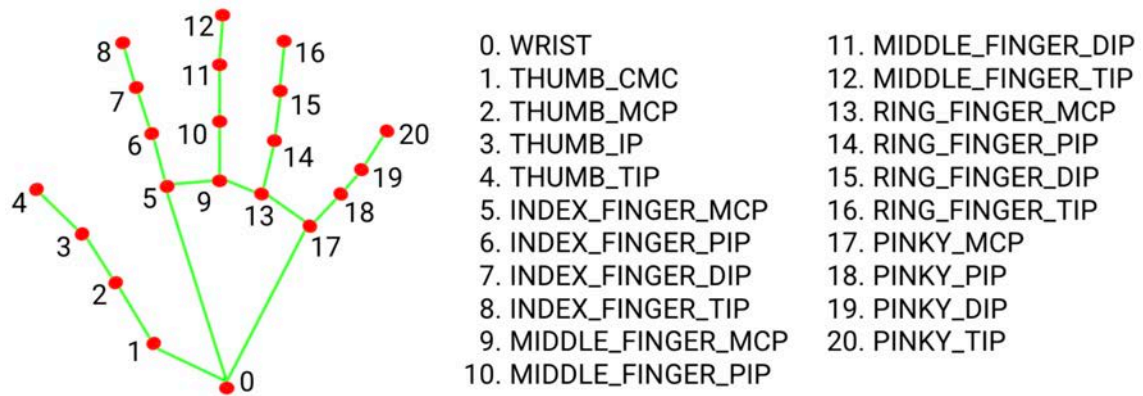


Figure 4.1: 21 hand keypoints in MediaPipe. (Figure source: [8]).



Figure 4.2: Example of using MediaPipe hand landmark model. (Figure source: [8]).

it uses 2x21-keypoint hand keypoint estimation [2].

4.1.3 MMPose

MMPose [28] is a PyTorch-based pose estimation open-source toolkit, a member of the OpenMMLab Project. It contains a rich set of algorithms for 2D multi-person human pose estimation, 2D hand pose estimation, 2D face landmark detection, 133 keypoint whole-body human pose estimation, fashion landmark detection, and animal pose estimation, as well as related components and modules. It consists of 8 components, offering among others an API, datasets, and models

4.2 Models Comparison

Several frameworks are available for detecting and localizing hand keypoints in images, with the most commonly used being MediaPipe and OpenPose. These frameworks differ primarily in their approach to keypoint detection. MediaPipe follows a top-down approach, first detecting the hand as a whole and then identifying its individual keypoints. In contrast,

OpenPose employs a bottom-up approach, initially detecting individual keypoints and then grouping them to form a complete hand.

One key consequence of this difference is how confidence metrics are assigned. MediaPipe provides a global confidence score for the entire hand, helping to determine whether the detected structure is indeed a hand. OpenPose, on the other hand, assigns individual confidence scores to each keypoint, which may offer finer control in some applications but lacks an overall reliability measure for the entire hand. Another significant distinction is that MediaPipe outputs 3D keypoints in the (x, y, z) format by estimating depth using inverse kinematics, whereas OpenPose only provides 2D keypoints.

In reviewing related literature, we found that most researchers opt for MediaPipe or OpenPose, while MMPose is rarely used. Furthermore, obtaining a license for MMPose is more complex, making it a less attractive choice for many projects. Given these factors, we chose to work with MediaPipe, as its top-down approach, built-in depth estimation, and straightforward implementation align well with the needs of our method.

4.3 Our Approach

The keypoint coordinates provided by MediaPipe are normalized. The x and y coordinates are scaled between 0 and 1, and we denormalize them by multiplying with the image width and height to obtain pixel values corresponding to the original input image. The z coordinate, on the other hand, is normalized relative to the wrist (keypoint 0), meaning that a value of 0 represents the wrist depth, while other values indicate depth differences relative to it. However, since the magnitude of z uses roughly the same scale as x [8], we again use the width of the image to de-normalize it although it may lead to some inaccuracies in the depth of the hand.

To assess the accuracy of the detected keypoints, we have developed custom visualization functions, allowing us to manually review the encoder's performance using both normalized and de-normalized values. Additionally, we have implemented a live version of our system, utilizing the computer's camera to process real-time hand keypoint detection. This implementation leverages specialized MediaPipe functions that, if our optimization pipeline runs efficiently, could allow the previously estimated 3D hand pose to serve as an initialization for the next optimization step.

Beyond MediaPipe, our implementation also integrates OpenCV [29], an open-source library widely used for machine learning, image processing, and computer vision tasks. OpenCV aids in handling image operations, rendering visualizations, and managing real-time processing within our framework.

4.4 The ReRun Viewer

Effective visualization plays a crucial role in the testing and development of our project. After exploring various alternatives, we chose to utilize ReRun Viewer [30] for its flexibility and real-time data streaming capabilities.

ReRun is designed for multimodal data processing, enabling users to model, ingest, store, query, and visualize robotics-style data. It is widely applied in fields such as robotics, spatial AI, industrial processing, simulation, and health. Using the ReRun SDK, we can log different data types, including images, tensors, point clouds, and text. These logs can then be streamed live to the ReRun Viewer or saved for later analysis [31].

In our implementation, we log hand keypoints as 3D points, assigning distinct colors to differentiate between the ground truth and predicted values. Each keypoint is labeled with a unique identifier (0–20), allowing us to track changes over time. This visualization helps us analyze the optimization process across iterations, as well as revisit previous steps if needed. Additionally, once the final MANO parameters are obtained, we can render and visualize the 3D hand mesh within ReRun Viewer. Additionally, we also log the 2D input image alongside the 3D keypoints. In the 2D space of the image, we also log the resulting hand mesh as well as the keypoints of the MediaPipe and the keypoints of the final MANO hand to see if they match. Examples of our ReRun environment can be seen later in Chapter 7.

Chapter 5

Optimization - Theoretical Background

Optimization is the mathematical process of finding the best values for input parameters that minimize or maximize an objective function. In the context of hand pose estimation, the goal is to minimize a loss function that quantifies the discrepancy between predicted hand keypoints and ground truth data.

5.1 Optimization Methods

Optimization methods can generally be categorized into first-order and second-order methods. First-order methods rely on the gradient, which is the vector of first derivatives of the objective function. This gradient provides information about the local slope, enabling algorithms such as gradient descent to iteratively adjust the parameters in the direction of steepest descent [32]. These methods are widely used due to their simplicity and efficiency in high-dimensional optimization problems.

Second-order methods, on the other hand, utilize the second derivative (the derivative of the derivative), which describes how the gradient itself changes. The second derivative of a multivariate function is captured in the Hessian matrix, a square matrix containing all second-order partial derivatives of the objective function [32]. Second-order methods, such as Newton's method, leverage the Hessian to determine not only the optimal direction of movement but also an appropriate step size, improving convergence efficiency and stability compared to first-order approaches [32, 33].

Quasi-Newton optimization algorithms, including BFGS (Broyden-Fletcher-Goldfarb-Shanno) and L-BFGS (Limited-memory BFGS), approximate the Hessian matrix to achieve

second-order-like performance without directly computing the full Hessian, making them more practical for large-scale optimization problems. These methods will be discussed in detail in the following sections.

5.1.1 BFGS Algorithm

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm is an iterative optimization method used to solve unconstrained optimization problems [34]. Given a nonlinear objective function:

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \quad (5.1)$$

where $f: \mathbb{R}^n \rightarrow \mathbb{R}$, the algorithm iteratively updates the estimate $\mathbf{x}_k$ to find the local minimum.

Algorithm Steps

1. **Initialization:** Start with an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$ and an initial Hessian approximation $B_0 = I$.

2. **Compute search direction $\mathbf{p}_k$** by solving:

$$B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k). \quad (5.2)$$

3. **Line search:** Find a step size α_k that approximately minimizes:

$$\alpha_k = \arg \min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k). \quad (5.3)$$

4. **Update position:**

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k. \quad (5.4)$$

5. **Compute differences:**

$$\mathbf{s}_k = \alpha_k \mathbf{p}_k, \quad (5.5)$$

$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k). \quad (5.6)$$

6. **Update Hessian approximation** using:

$$B_{k+1} = B_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{B_k \mathbf{s}_k \mathbf{s}_k^T B_k}{\mathbf{s}_k^T B_k \mathbf{s}_k}. \quad (5.7)$$

7. **Convergence check:** If $\|\nabla f(\mathbf{x}_k)\| \leq \epsilon$, stop.

Inverse Hessian Update

Instead of updating B_k , the inverse Hessian approximation $H_k = B_k^{-1}$ can be updated using the Sherman–Morrison formula [34]:

$$H_{k+1} = H_k + \frac{(\mathbf{s}_k^T \mathbf{y}_k + \mathbf{y}_k^T H_k \mathbf{y}_k)(\mathbf{s}_k \mathbf{s}_k^T)}{(\mathbf{s}_k^T \mathbf{y}_k)^2} - \frac{H_k \mathbf{y}_k \mathbf{s}_k^T + \mathbf{s}_k \mathbf{y}_k^T H_k}{\mathbf{s}_k^T \mathbf{y}_k}. \quad (5.8)$$

5.1.2 Limited-memory BFGS (L-BFGS)

Limited-memory BFGS (L-BFGS)[35] is a Quasi-Newton optimization method that builds upon the BFGS algorithm while significantly reducing memory usage. It is a limited-memory version of BFGS, which stores only a few vectors to approximate the inverse Hessian, making it more efficient for large-scale problems, such as parameter estimation in machine learning.

Similar to BFGS, L-BFGS seeks to minimize a differentiable function $f(\mathbf{x})$ over an unconstrained vector $\mathbf{x} \in \mathbb{R}^n$. Instead of storing and updating a full $n \times n$ approximation of the inverse Hessian matrix, L-BFGS maintains a compact history of the last m updates to the position $\mathbf{x}$ and gradient $\nabla f(\mathbf{x})$. Typically, the history size m is kept small (often $m < 10$), which significantly reduces computational and storage requirements.

The advantage of this approach is that it allows L-BFGS to approximate second-order information without explicitly constructing the inverse Hessian. Instead, it performs the necessary vector transformations implicitly, ensuring efficiency even for problems with a large number of variables.

5.2 Loss Functions

Loss functions play a crucial role in optimization problems, particularly in machine learning and statistical modeling. They measure the discrepancy between predicted and actual values, guiding optimization algorithms to minimize this error.

5.2.1 Mean Squared Error

The Mean Squared Error (MSE) [36] loss is one of the most commonly used loss functions for regression problems. It quantifies the average squared difference between the predicted

values $\hat{y}_i$ and the true values y_i :

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (5.9)$$

where n is the number of data points.

MSE is widely used because:

- It penalizes large errors more than small ones due to the squaring operation.
- It provides a smooth, differentiable function, making it suitable for gradient-based optimization methods.
- It assumes a Gaussian noise distribution in the data, which aligns well with many real-world scenarios.

However, a key drawback of MSE is its sensitivity to outliers. Since errors are squared, large deviations have a disproportionately high impact on the overall loss, which can lead to models that prioritize minimizing extreme errors rather than general performance.

Despite this limitation, MSE remains a fundamental choice for regression tasks, especially when data is normally distributed and outliers are not a significant concern.

5.2.2 Geman-McClure

The Geman-McClure (GM) loss function is a robust alternative to the MSE loss, designed to mitigate the influence of outliers. Originally introduced in the context of tomographic image reconstruction [37], the GM loss has since been applied in various domains, including computer vision and 3D human pose estimation [38].

The GM loss function is defined as:

$$L_{\text{GM}}(r) = \frac{\rho^2 r^2}{r^2 + \rho^2} \quad (5.10)$$

where r represents the residual error (*i.e.*, the difference between predicted and true values), and ρ is a parameter that controls the sensitivity of the function to large residuals.

- For small residuals ($r \ll \rho$), the loss behaves similarly to the squared error.
- For large residuals ($r \gg \rho$), the loss asymptotically approaches ρ^2 , limiting the impact of outliers.

This property makes GM loss more robust compared to MSE, as it prevents large errors from dominating the overall loss. It has been particularly useful in applications where noisy detections need to be down-weighted, such as in 3D human body reconstruction [38].

Given its robustness, GM loss serves as an effective alternative to traditional MSE loss in scenarios where the presence of outliers can significantly affect model performance.

5.2.3 Huber

Huber loss is a robust loss function that is less sensitive to outliers compared to the squared error loss. It is widely used in robust regression and can also be adapted for classification tasks. The function was first introduced by Huber in [39] and is particularly useful in scenarios where a balance between MSE and Mean Absolute Error (MAE) is desired.

Mathematically, the Huber loss function is defined as:

$$L_{\delta}(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta (|a| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \quad (5.11)$$

where a represents the residual error, *i.e.*, the difference between the observed and predicted values:

$$a = y - f(x). \quad (5.12)$$

- For small residuals ($|a| \leq \delta$), the loss behaves like the squared error, ensuring smooth differentiation.
- For large residuals ($|a| > \delta$), the loss transitions to an absolute error formulation, reducing the impact of outliers.

The function is quadratic for small residuals and linear for larger residuals, ensuring continuity at $|a| = \delta$. This hybrid approach makes Huber loss more robust than MSE while avoiding the non-differentiability of MAE at zero.

5.2.4 Anatomical Priors

Let us now introduce a more specific loss constraint that is directly related to hand articulation and our problem. The anatomical joint limits error E_{limits} is a penalty term applied to

the 45-dimensional pose vector, where each joint has three DoF. This error function ensures that estimated hand poses remain within experimentally determined anatomical constraints, thereby enforcing plausible hand articulations. Similar to [21], this loss is implemented as a soft constraint using exponential functions that activate when joint angles exceed predefined limits:

$$E_{\text{limits}}(\boldsymbol{\theta}) = a_{\text{limits}} \sum_{i=0}^m (e^{l_i - \theta_i} + e^{\theta_i - u_i}), \quad (5.13)$$

where $[l_i, u_i]$ denote the lower and upper bounds for joint angle θ_i , and a_{limits} is an experimentally determined weight factor. The exponential terms enforce smooth constraints that discourage hand poses outside of the allowable range.

The concept of anatomical constraints for hand motion was first introduced by Lin *et al.* [9] and has since been adopted by various works, including [21], [40], and others. These constraints are primarily classified as “Type I constraints”, which limit the natural range of motion of each finger without applying external forces. For example, the flexion and abduction/adduction ranges of motion are typically constrained as follows:

$$\begin{aligned} 0^\circ \leq \theta_{MCP_F} \leq 90^\circ, \quad 0^\circ \leq \theta_{PIP_F} \leq 110^\circ, \\ 0^\circ \leq \theta_{DIP_F} \leq 90^\circ, \quad -15^\circ \leq \theta_{MCP_{AA}} \leq 15^\circ, \\ \theta_{MCP_{AA}} = 0^\circ, \quad \theta_{TM_{AA}} = 0^\circ. \end{aligned} \quad (5.14)$$

where θ_{MCP} , θ_{PIP} , θ_{DIP} , and θ_{TM} refer to the joints shown in Figure 5.1. The subscript F denotes flexion, while AA denotes abduction/adduction motion. A commonly adopted constraint states that the middle finger exhibits little abduction/adduction motion, effectively reducing one DoF from the 21-DoF hand model. Additionally, the thumb motion is approximated using four parameters instead of five. The DIP, PIP, and MCP joints of the index, middle, ring, and little fingers are considered planar manipulators, meaning they move within a single plane since the DIP and PIP joints have only one DoF for flexion.

By incorporating these anatomical constraints, we improve the realism and feasibility of estimated hand poses, preventing unnatural articulations. This loss is a crucial component in biomechanical hand modeling.

5.3 Implementation Details

For optimization, we employed the BFGS and L-BFGS algorithms from `scipy.minimize` [41], as they are well-suited for smooth, differentiable functions and provide efficient con-

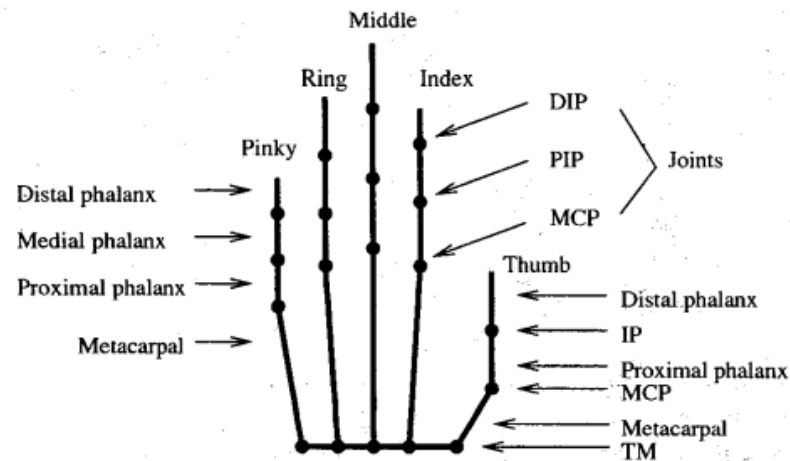


Figure 5.1: Kinematical structure and joint notations. (Figure source: [9]).

vergence properties.

Regarding loss functions, we utilized PyTorch's built-in MSELoss for Mean Squared Error computation. However, for the Geman-McClure and Huber loss functions, we implemented custom versions to ensure proper integration within our framework. The corresponding implementations can be found in the Appendix.

Additionally, anatomical constraints are inherently implemented in the Manotorch [23] framework, which was a key factor in our decision to adopt it. This allowed us to leverage its built-in constraints to ensure biomechanically plausible hand articulations without requiring custom constraint enforcement.

Chapter 6

Proposed Method

6.1 Overview

To tackle the problem of fitting an articulated 3D hand from a single RGB image, we design the pipeline shown in Figure 6.1. The input is a standard RGB image containing a human hand, and the output is a 3D hand in the exact same pose and orientation.

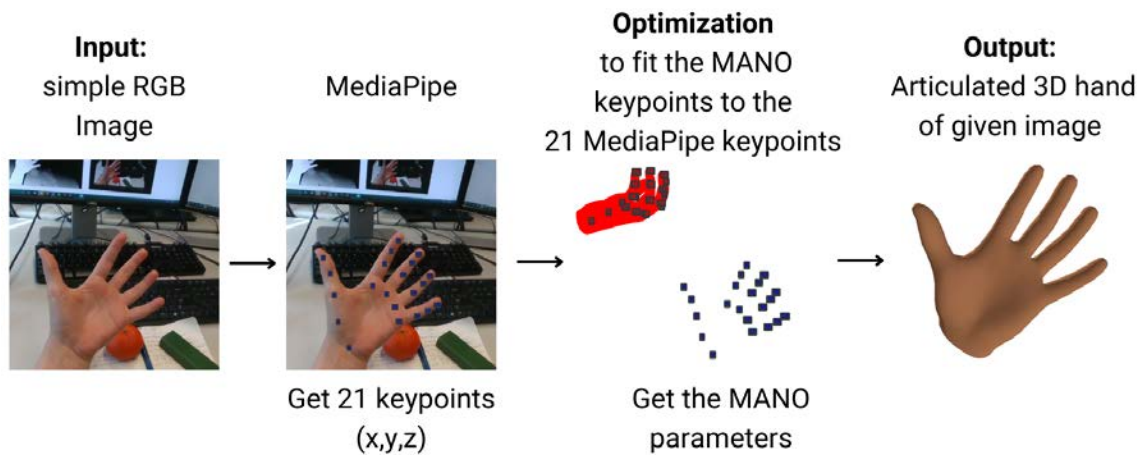


Figure 6.1: Our proposed pipeline

Since the input is a simple 2D image without depth information, just like images captured by conventional cameras, our approach allows real-world applicability and is easier to integrate (as it is generic) to relevant systems. The pipeline is designed to handle both left and right hands, as the estimator can correctly infer handedness and incorporate it into the later parameterized processing steps. However, our method currently does not support reconstructing multiple hands within the same image simultaneously.

The first step in the pipeline involves passing the input image through an estimator to extract the 21 hand keypoints along with handedness information. We utilize the MediaPipe Hand model [8] for this purpose. The keypoints are initially obtained in a normalized (x, y, z) format, which we then de-normalize using the image width and height (see Chapter 4). This process results in a list of keypoints corresponding to pixel locations in the input image, which we later use as ground truth parameters for fitting the MANO hand model [7].

For the optimization step, our goal is to fit the MANO model's keypoints to the extracted "ground truth" keypoints from MediaPipe. Given the pose and shape parameters of MANO, we can obtain the 21 hand joint locations through linear interpolation. MANO takes 45 pose parameters, 10 shape parameters, and 3 global rotation parameters as input to generate a 3D mesh that represents a unique hand configuration (see Chapter 3).

The fitting process begins with a neutral "zero" pose and shape. Through a series of transformations and iterative optimization, we adjust the MANO parameters to align its 21 keypoints with the estimated MediaPipe keypoints by minimizing a loss function. The details of the optimization process are discussed later. Once the optimization is complete, we obtain the final MANO parameters, which are then used to generate a 3D articulated mesh, producing the desired reconstruction.

6.2 Problems Encountered during Development

During the development of this pipeline, we faced several challenges.

In one of the initial versions of our implementation, we relied on a software framework [22] that provided MANO functionality, including functions to set MANO parameters and compute joint locations (keypoints) through linear interpolation. At this stage, our loss function was kept simple, defined as the MSE between the MANO keypoints and the "ground truth" keypoints extracted from MediaPipe.

One of the first issues we encountered was the mismatch in keypoint ordering between MANO and MediaPipe. MANO's joint order differed, particularly because fingertip keypoints were added later in its representation, whereas MediaPipe followed a more intuitive ordering. To address this, we had to manually reorder the keypoints to ensure alignment between the two systems.

After implementing this, we conducted tests and observed that while the method seemed

to work in some very specific cases, it completely failed in the rest of them. The most critical issue arose when the input hand was in a different global rotation than MANO's default pose. Our method struggled to converge, leading to highly inaccurate results.

The root of the problem was in the initialization: MANO starts in a neutral pose, shape, and rotation since we have no prior information about the input hand's orientation and initialized every parameter with zeros. However, if the hand in the input image had a significantly different rotation—for example, in a “handshake” position, where the wrist (keypoint 0) is at the top and the fingers point downward, opposite to MANO's default orientation—the optimization process failed, resulting in completely implausible hand meshes.

To mitigate this, we first attempted a two-stage optimization approach. In the first stage, we optimized only a few keypoints (or even just the wrist keypoint) to quickly estimate the global rotation of the hand. The second stage then used this result as an initialization and proceeded with the full keypoint-based optimization. We experimented with various combinations—different sets of keypoints, different loss functions, and different optimization techniques. However, this approach was ultimately unsuccessful, as optimizing for just a few keypoints often produced extreme deformations, leading to poor initializations for the second stage.

Another idea was to enforce anatomical constraints on the hand pose to ensure plausible reconstructions. This led us to switch to the Manotorch framework [23], which included a built-in anatomical constraints loss. However, even with these constraints in place, the fundamental problem remained unresolved.

We also explored alternative strategies, such as relying solely on the 2D projections of keypoints from both MediaPipe and MANO to simplify the optimization process. Additionally, we experimented with transforming the MediaPipe keypoints using the camera's intrinsic parameters before computing the loss. However, this approach introduced a dependency on camera calibration, making it less suitable for our intended method. Despite these efforts, none of the methods resolved the issue. Our next idea for properly initializing the optimization process and ensuring accurate results, was based on applying a rigid transformation, which we explored in the subsequent stages of development.

6.3 Theoretical Background on Rigid Transformation

Rigid transformation [42], in general, involves applying a rotation, translation, and optionally scaling to align two sets of points while preserving their internal geometric relationships. In our work, we leverage rigid transformation to align the extracted MediaPipe keypoints with the MANO model keypoints by correcting global rotation, translation, and scale discrepancies. This process allows for a more accurate initialization of the optimization step.

A rigid transformation consists of two main operations:

- Translation: A shift of an object in 3D space along the x , y , and z axes without altering its orientation.
- Rotation: A transformation that changes the orientation of an object while preserving its shape and size.

Mathematically, a rigid transformation of a point $\mathbf{p}$ in 3D space can be expressed as:

$$\mathbf{p}' = \mathbf{R} \cdot \mathbf{p} + \mathbf{t}, \quad (6.1)$$

where:

- $\mathbf{p}, \mathbf{p}' \in \mathbb{R}^3$ are the original and transformed points, respectively.
- $\mathbf{R} \in \mathbb{R}^{3 \times 3}$ is a rotation matrix that preserves distances and angles.
- $\mathbf{t} \in \mathbb{R}^3$ is a translation vector.

To represent both rotation and translation in a unified manner, we use a homogeneous transformation matrix:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4}, \quad (6.2)$$

where the bottom row $[0, 0, 0, 1]$ ensures that the transformation can be applied using homogeneous coordinates.

Using this representation, a 3D point $\mathbf{p}$ can be expressed in homogeneous coordinates as $\mathbf{p}^H = [x, y, z, 1]^T$, and its transformation is computed as:

$$\mathbf{p}'^H = \mathbf{T} \cdot \mathbf{p}^H. \quad (6.3)$$

6.3.1 Computing the Optimal Rigid Transformation

Given two sets of corresponding 3D keypoints, $\mathbf{S} = \{s_1, s_2, \dots, s_n\}$ (source points) and $\mathbf{T} = \{t_1, t_2, \dots, t_n\}$ (target points), our goal is to find the optimal rigid transformation that minimizes the difference between them.

This can be achieved using the “Singular Value Decomposition” (SVD) [43] approach:

1. Compute the centroids of both sets:

$$\mathbf{s}_c = \frac{1}{n} \sum_{i=1}^n s_i, \quad \mathbf{t}_c = \frac{1}{n} \sum_{i=1}^n t_i. \quad (6.4)$$

2. Center the keypoints by subtracting their centroids:

$$\mathbf{S}' = \mathbf{S} - \mathbf{s}_c, \quad \mathbf{T}' = \mathbf{T} - \mathbf{t}_c. \quad (6.5)$$

3. Compute the covariance matrix:

$$\mathbf{H} = \mathbf{S}'^T \cdot \mathbf{T}'. \quad (6.6)$$

4. Perform SVD:

$$\mathbf{H} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T. \quad (6.7)$$

5. Compute the optimal rotation matrix:

$$\mathbf{R} = \mathbf{V} \cdot \mathbf{U}^T. \quad (6.8)$$

If $\det(\mathbf{R}) < 0$, reflection occurred, and we correct it by flipping the last column of $\mathbf{V}$:

$$V_{[:, -1]} = -V_{[:, -1]}. \quad (6.9)$$

6. Compute the optimal translation vector:

$$\mathbf{t} = \mathbf{t}_c - \mathbf{R} \cdot \mathbf{s}_c. \quad (6.10)$$

Our custom implementation in python can be found in Appendix.

6.4 Description of the Optimization Pipeline

In designing and developing this final version of our optimization pipeline, our primary objective was to address the “handshake” problem and resolve issues related to global rotation and translation during initialization. To achieve this, we implemented a rigid transformation to align the two sets of keypoints: the “target” keypoints, obtained from MediaPipe’s 3D predictions, and the “MANO” keypoints, computed using the MANO hand model in its neutral configuration (zero shape, pose, and root pose).

Additionally, we applied scaling to ensure both sets of keypoints were properly aligned before beginning the optimization process. This preprocessing step significantly improved convergence and accuracy. The complete optimization pipeline is illustrated in Figure 6.2 and will be thoroughly described in this section.

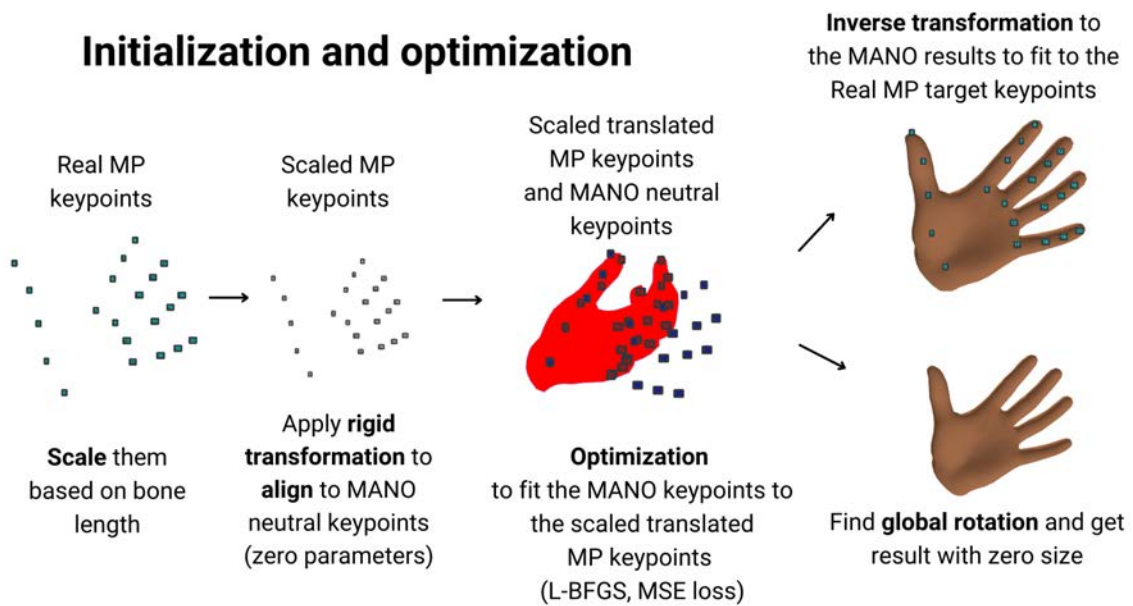


Figure 6.2: Initialization and optimization pipeline

6.4.1 Extracting Key Features

The first step in our optimization pipeline involved extracting the six main palm joints from the target keypoints and the MANO model, namely keypoints [0,1,5,9,13,17] as illustrated in Figure 4.1. These specific points were chosen because they form the stable structure of the palm, minimizing the influence of finger articulation. We intentionally excluded key-point 2, *i.e.* the MCP of the thumb, as it exhibits more variability and is not considered part

of the rigid palm structure. The rigid transformation was computed using these palm joints, ensuring that alignment would primarily focus on the central structure of the hand rather than being skewed by finger positions. By focusing on the palm, we improved the stability of the transformation and reduced outliers' influence.

6.4.2 **Scaling and Aligning. Rigid Transformation as initialization**

To properly initialize the optimization process, we applied a rigid transformation to align the “target” keypoints (from MediaPipe) with the MANO keypoints. This transformation corrected global rotation, translation, and scale, ensuring that both sets of keypoints were well-aligned while preserving their internal geometric relationships.

To compute the transformation, we implemented a custom function. However, alternative solutions exist, such as the Open3D `compute_transformation` function [44], which estimates the optimal rotation matrix and translation vector by aligning the centroids of the two sets of points. Our function returns a 4×4 transformation matrix, where the top-left 3×3 block represents the rotation, the top-right 3×1 column is the translation vector, and the bottom row is used for homogenous coordinates. For scaling, we computed a scale factor using the distances between keypoints 0 (wrist) and 5 (INDEX FINGER MCP) in both the “target” and MANO keypoints. This ensured that the scaling was proportional to the anatomical structure of the hand, preserving the relative proportions while aligning the sizes of the two sets. By incorporating this transformation, we minimized the gap between the two sets of keypoints, providing a robust initialization for the optimization process.

6.4.3 **Optimization**

After the initial alignment, we used `scipy.minimize` [41] to optimize the MANO hand model parameters, specifically leveraging the L-BFGS method. This choice was driven by its efficiency and suitability for high-dimensional optimization problems. Although BFGS was also a viable option in our implementation, it proved to be more computationally expensive, making it less practical for large-scale evaluations.

We also experimented with alternative optimization methods, such as the least-squares Dogbox method [45] and others. While these approaches demonstrated adequate performance in certain cases, they failed to converge reliably across a broader dataset, preventing a thorough quantitative evaluation. Despite initial qualitative assessments suggesting that these

methods would yield inferior results, their inconsistency further reinforced our decision to prioritize L-BFGS-B.

The optimization process aimed to minimize the loss function between the transformed MANO keypoints and the target keypoints. By initializing the MANO parameters with the results of the rigid transformation, we ensured that the optimization began from a reasonable starting point, reducing the risk of convergence to poor local minima resulting in the “handshake” problem or similar other ones.

6.4.4 Loss functions

The loss functions employed in our optimization process are analyzed in detail in Chapter 5. We primarily experimented with three loss functions: MSE, GM [7], and Huber loss.

To improve the accuracy of fingertip alignment, we explored weighted loss functions, as we observed that most keypoints were densely concentrated around the palm, which was already well-aligned. Due to the nature of the loss functions, the optimization process primarily focused on minimizing the error in these denser regions, often leading to less precise alignment of the fingertips. However, in real-world hand movement, fingertips play a critical role in defining hand gestures and poses, making their accurate positioning essential. To address this, we applied weighted loss functions across all three variations (MSE, GM, and Huber), giving higher importance to fingertip keypoints to ensure their proper alignment.

Additionally, we attempted to integrate anatomical joint constraints into the loss function to enforce physically plausible hand poses. However, this approach proved ineffective, as it restricted the optimization process too severely. Despite experimenting with various weighting schemes, the results did not improve. Instead, we adopted a two-stage optimization strategy:

- Stage 1: A standard MSE loss function was used to obtain an initial estimate of the hand pose.
- Stage 2: The output of Stage 1 was refined using anatomical loss constraints, combined with an MSE loss applied only to 2D keypoints. This ensured that the estimated hand remained within realistic anatomical bounds.

As reported in the evaluation chapter (Chapter 7), this two-stage approach resulted in worse overall metrics but successfully corrected depth-related errors caused by MediaPipe’s

3D predictions in specific cases.

Lastly, of course it is also important to use the right tolerance and weights in the loss function to prevent overfitting where the results of the mesh do not represent a plausible hand.

6.4.5 Reversing Transformation and Visualization

After optimization, we reversed the rigid transformation to recover the MANO keypoints and mesh in the original scale, orientation, and position. The rotation matrix, translation vector, and scaling factor used for initialization were inverted to map the optimized MANO results back to the target's coordinate system so that we can inspect our results visually.

To determine the root pose, we computed the axis-angle representation of the rotation matrix derived from the rigid transformation and incorporated it into the MANO model's parameters. This ensured that the global rotation was accurately represented in the final output. This means that we now also have a 3D hand with the correct global orientation and pose but with the zero shape that is not really a problem.

6.4.6 Using Images in the Wild

With the improvements in initialization and optimization, the model is now robust enough to handle hand keypoints extracted from images captured “in the wild,” such as casual or uncontrolled environments. Moreover, our current implementation does not use the camera parameters so every 2D image containing a hand can be used as an input. The rigid transformation enables the model to adapt to a wide range of global poses and scales, while correctly initialized optimization ensures an accurate fit to the target keypoints. Therefore, these changes allow for realistic hand reconstructions, even from noisy or low-resolution input data, making the method practical for real-world applications.

Chapter 7

Evaluation and Results

7.1 Quantitative Evaluation

For the quantitative evaluation of our method, we follow the evaluation pipeline proposed in [21] and [46]. This ensures direct comparability between our approach and both their methods, as well as with other SotA techniques they benchmarked.

Although [21] employs an optimization-based approach similar to ours, while [46] follows a learning-based approach, we evaluate our method against both SotA deep learning methods to provide a comprehensive performance assessment.

7.1.1 Metrics

To ensure a direct and fair comparison with SotA methods, we evaluate our approach using the same metrics as [21] and [46]. These metrics assess the accuracy of 3D hand pose estimation by measuring the deviation of predicted keypoints from the ground truth.

We compute the Root Mean Square Error (RMSE), also referred to as End-Point Error (EPE), for all 3D hand joints. This metric quantifies the absolute difference between the estimated keypoints and the ground truth in millimeters. A lower RMSE value indicates a more precise reconstruction of the hand pose.

The Percentage of Correct Keypoints (PCK) evaluates the proportion of keypoints that are correctly estimated within a given threshold distance from the ground truth. We compute PCK for thresholds ranging from 20mm to 50mm to analyze accuracy at different tolerance levels. To summarize the PCK performance across different thresholds, we also compute the Area Under the Curve (AUC) for this range. A higher AUC value indicates better overall accuracy,

with an ideal score reaching 1.0, representing a perfect fit. This metric is particularly useful for comparing methods holistically, as it accounts for performance across multiple threshold levels rather than relying on a single fixed distance.

7.1.2 Data

Table 7.1 presents the most recent datasets with optical data that are used for the task of hand pose estimation. These either contain only RGB images or are combined with depth data, while the data are synthetic or real. Moreover, we know if there is object interaction of the hands in the frames, the number of joints annotated and if the view is egocentric or third-person. Lastly, we can see the number of subjects (if known) that were scanned for the creation of the dataset. Additionally, Table 7.2 contains information about the number of frames of each dataset and if they are somehow previously split into training, validation and test sets, as well as the conference and year of their introduction.

Table 7.1: A collection of datasets comprising optical annotated data for hand pose estimation. “**#J**” is the number of joints annotated, and “**Obj int**” states whether there is object interaction.

Dataset	Type	Synthetic	Obj int	#J	View	#Subs
HIU-DMTL-Data [20, 47]	RGB	✓ ✓(and real)	no	21	3rd/ego	200
InterHand2.6M [48]			no	21	3rd	27
YouTube 3D Hands [19]			yes	-	3rd	-
OneHand10K [49]			no	21	3rd	1
FreiHAND [14]			yes	21	3rd	-
GANerated Hands [16]			both	21	ego	-
CMU Panoptic HandDB [2]			no	21	3rd	-
MHP [50]			no	21	3rd	9
MVHand [51]	RGB+D	✓ ✓ ✓ ✓	no	21	3rd	4
ContactPose [52]			yes	21	3rd	50
Ego3DHands [53]			no	21	ego	1
ObMan [54]			yes	-	-	-
EgoDexter [11]			yes	5	ego	4
SynthHands [11]			both	21	ego	2
RHD [13]			no	21	3rd	20
STB [55]			no	21	3rd	1
Dexter+Object [12]			yes	5	3rd	2
UCI-EGO [56]			no	26	ego	2
Dexter1 [57]			no	6	3rd	1

Table 7.2: Additional information of the datasets.

Dataset	Type	#Frames	Presented
HIU-DMTL-Data [20, 47]	RGB	40	ICCV 2021
InterHand2.6M [48]		2.6M	ECCV 2020
YouTube 3D Hands [19]		47k/1.5k/1.5k	CVPR 2020
OneHand10K [49]		10k/1.3k	TCSVT 2019
FreiHAND [14]		130k/396	ICCV 2019
GANerated Hands [16]		330k	CVPR 2018
CMU Panoptic HandDB [2]		14,817	CVPR 2017
MHP [50]		80k	IVC 2017
MVHand [51]	RGB+D	83k	BMVC 2021
ContactPose [52]		2.9M	ECCV 2020
Ego3DHands [53]		50k/5k	arXiv 2020
ObMan [54]		150k	CVPR 2019
EgoDexter [11]		1485	ICCV 2017
SynthHands [11]		63,53	ICCV 2017
RHD [13]		41k/2.7k	ICCV 2017
STB [55]		18k	ICIP 2017
Dexter+Object [12]		3,014	ECCV 2016
UCI-EGO [56]		400	ECCVW 2014
Dexter1 [57]		2,137	ICCV 2013

Since our method does not involve a training phase, we exclusively utilize datasets for evaluation purposes. We selected the EgoDexter [11] and Dexter+Object [12] datasets based on several criteria. First, these datasets have been widely used in previous studies, allowing for a direct comparison of our method with existing approaches. Second, they provide a sufficient number of frames for robust evaluation, while maintaining a manageable dataset size. Additionally, both datasets include 3D annotated data, enabling us to effectively validate the accuracy of our 3D hand fitting. Finally, among the various benchmark datasets, EgoDexter and Dexter+Object provide the most unbiased test environment, as no model of the ones we use to compare our method has used them for training. As noted in [46], this makes them ideal for assessing generalization performance across different methods.

EgoDexter [11]

EgoDexter is an RGB-D dataset designed for evaluating hand-tracking algorithms in cluttered environments with significant occlusions. It consists of four video sequences featuring four different actors (two female) interacting with various objects in diverse settings, including a living room, office, and kitchen. The dataset was collected using an Intel RealSense SR300 camera, providing synchronized RGB and depth data at a resolution of 640×480 pixels.

Ground truth annotations include manually labeled 3D fingertip positions on depth data. However, only fingertips are annotated, and due to occlusions, not all frames contain annotations for all five fingertips. In total, 1,485 out of 3,190 frames include fingertip annotations.

Dexter+Object [12]

Dexter+Object is an RGB-D dataset designed for evaluating algorithms that track both hands and objects simultaneously. It comprises six video sequences featuring two actors (one female) interacting with a simple cuboid-shaped object. The dataset was captured using a Creative Senz3D camera, which provides synchronized RGB and depth data.

Ground truth annotations include manually labeled 3D fingertip positions and three cuboid corners on depth data. However, for our evaluation, we only utilize the five fingertip positions. While all 3,014 frames contain annotations, occlusions are present, particularly in sequences where the cuboid obstructs parts of the hand.

7.1.3 Preprocessing

Both datasets provide manually annotated 3D hand keypoints in millimeters in camera coordinates. However, our method, which is based on MediaPipe's predictions, outputs 21 keypoints in pixel space in (x, y, z) format, including depth values. Thus, we must transform them to camera coordinates to ensure compatibility with the dataset's ground truth values. This introduces two key preprocessing steps:

1. Extracting only the relevant keypoints, as the datasets provide only up to five fingertip keypoints per frame, so we must filter out the corresponding keypoints from our predictions. Also let us note that since both datasets include many frames with occluded hands, not all frames contain 5 fingertip keypoints since they are manually annotated, and when the fingertips are not visible due to occlusion there are not annotated. This means that some frames may have from 0 to 5 keypoints so we also need to extract the correct corresponding keypoints from our results each time.
2. Transforming pixel-space predictions into camera coordinates. This requires utilizing the intrinsic camera parameters (such as focal length) provided in the datasets.

Handling Depth Differences

Before transforming the pixel-space coordinates into millimeters, we first address the depth (Z-axis) differences between the datasets and MediaPipe. In the Dexter+Object and EgoDexter datasets, depth values are measured from the camera sensor outward, meaning that 0mm corresponds to the camera sensor. However, in MediaPipe, depth is defined relative to the wrist (keypoint 0), meaning our predicted depth values differ significantly from the dataset's convention.

To correct this, we apply a Z-axis translation that aligns the depth values of our predictions with the dataset's scale. This is achieved by computing the mean Z-value of the ground truth and predicted fingertip keypoints and shifting our predictions accordingly:

$$Z' = Z + (\text{mean}_Z^{\text{GT}} - \text{mean}_Z^{\text{pred}}) \quad (7.1)$$

This translation ensures that our predictions match the dataset's depth conventions, preventing errors in evaluation.

Transforming Pixel-Space Predictions to Camera Coordinates

Once the depth correction is applied, we transform our pixel-space predictions into camera coordinates using the camera intrinsics. Since depth values remain the same before and after the transformation, we can use them directly in the following equations:

$$X_c = \frac{(X_p - c_x) \cdot Z_c}{f_x}, \quad Y_c = \frac{(Y_p - c_y) \cdot Z_c}{f_y} \quad (7.2)$$

where:

- (X_p, Y_p) are pixel-space coordinates,
- (X_c, Y_c, Z_c) are camera-space coordinates,
- Z_c is the depth value, which remains unchanged in pixel-space and camera-space,
- (c_x, c_y) is the principal point (optical center),
- (f_x, f_y) are focal lengths in pixels.

This transformation ensures that our predictions are expressed in millimeters, rather than pixel units, enabling direct comparison with the dataset's ground truth values.

Final Global Alignment

Following previous work [46], we apply a final centroid alignment to better measure the local hand pose. We adjust the translation by shifting the predicted fingertip keypoints so that their centroid exactly matches the centroid of the ground truth keypoints.

$$T = \text{centroid}^{\text{GT}} - \text{centroid}^{\text{pred}} \quad (7.3)$$

where:

- $\text{centroid}^{\text{GT}} = \frac{1}{N} \sum_{i=1}^N \mathbf{p}_i^{\text{GT}}$ is the mean position of the ground truth fingertips.
- $\text{centroid}^{\text{pred}} = \frac{1}{N} \sum_{i=1}^N \mathbf{p}_i^{\text{pred}}$ is the mean position of the predicted fingertips.
- $\mathbf{p}_i = (X_i, Y_i, Z_i)$ are the 3D keypoint coordinates.
- N is the number of fingertip keypoints used (up to 5).

We then apply this translation shift to all predicted keypoints:

$$\mathbf{p}_i^{\text{aligned}} = \mathbf{p}_i^{\text{pred}} + T \quad (7.4)$$

for each fingertip $i = 1, \dots, N$.

This transformation modifies only the translation component, applying a simple vector shift T to the predicted keypoints. Importantly, rotation and scale remain unchanged, ensuring that the relative hand pose is fully preserved. Since this step does not alter the shape or orientation of the hand, it allows for a direct comparison by aligning keypoints in absolute space. By performing this final alignment, we effectively remove global positional differences, enabling a more accurate evaluation that focuses solely on how well the predicted hand pose matches the ground truth.

By following these preprocessing steps, we ensure a fair and accurate comparison of our method with other approaches.

7.1.4 Ablation Study

Table 7.3 presents the different experimental configurations we tested to evaluate our method on two datasets. Each experiment is identified by a letter (ID) and corresponds to a specific combination of optimizers and loss functions. For instance, in Experiment A, we employ the L-BFGS optimizer while minimizing a weighted combination of MSE loss for all 21 keypoints and an additional MSE loss specifically for the fingertips.

While we tested all loss function combinations using the L-BFGS optimizer, we selectively tested what we considered the most promising loss function setup with the BFGS optimizer. As also stated in Chapter 6, we also experimented with alternative optimization methods, such as the least-squares Dogbox method [45] and others. While these approaches demonstrated adequate performance in certain cases, they failed to converge reliably across a broader dataset, preventing a thorough quantitative evaluation. This explains why no other optimizers appear in the table.

Finally, in experiment H we adopted a two-stage optimization strategy:

- Stage 1: A standard MSE loss function was used to obtain an initial estimate of the hand pose.
- Stage 2: The output of Stage 1 was refined using anatomical loss constraints, combined

Table 7.3: The description of what combination of optimizer and loss function each experiment uses

ID	OPTIMIZERS		LOSSES						
	LBFGS	BFGS	MSE	MSE fingertips	Geman- McClure	Geman- McClure fingertips	Huber	Huber fingertips	Anatomical (2 stages)
A	✓		✓	✓					
B	✓		✓						
C	✓				✓	✓			
D	✓				✓				
E	✓						✓	✓	
F	✓						✓		
G		✓	✓	✓					
H	✓		✓						✓

with an MSE loss applied only to 2D keypoints. This ensured that the estimated hand remained within realistic anatomical bounds.

Table 7.4 presents the quantitative results of our experiments, reporting the End-Point Error (in mm) and the AUC of PCK for both datasets. Our findings indicate that most experiments perform consistently well, with results comparable to each other and to SotA methods, as discussed later.

This consistency is expected, as the different loss functions we experimented with are all variations of the MSE loss, designed to improve robustness against outliers. The best-performing results for each metric are highlighted in bold, although in many cases, multiple configurations achieve similar results.

Our observations suggest that weighted fingertips loss functions tend to yield slightly better performance across metrics. However, the standard loss functions also achieve competitive results, while the better results of the weighted fingertips loss may be attributed to the specific evaluation datasets focusing primarily on fingertip keypoints.

Additionally, we can observe that in experiment H, this two-stage approach integrating anatomical constraints resulted in worse overall metrics although it successfully corrected depth-related errors caused by MediaPipe’s 3D predictions in specific cases.

Overall, our results suggest that the most effective optimizer-loss combinations were Experiment A and Experiment G. This outcome aligns with expectations, as L-BFGS and BFGS

Table 7.4: Ablation Study: End-Point Error (mm) ($\downarrow$) and AUC of PCK ($\uparrow$) results of EgoDexter and Dexter+Object.

	EgoDexter		Dexter+Object	
ID	EPE (mm) ($\downarrow$)	AUC of PCK ($\uparrow$)	EPE (mm) ($\downarrow$)	AUC of PCK ($\uparrow$)
A	17.724	0.883	13.985	0.946
B	19.642	0.859	14.859	0.943
C	17.729	0.883	13.980	0.946
D	19.648	0.859	14.878	0.942
E	17.864	0.882	13.978	0.946
F	19.963	0.855	15.279	0.939
G	17.787	0.883	13.975	0.946
H	42.833	0.492	24.574	0.784

are closely related, with L-BFGS being a memory-efficient variant. If computational efficiency is considered alongside accuracy, Experiment A—using L-BFGS with a weighted combination of MSE losses—emerges as the optimal choice.

Figures 7.1 and 7.2 compare the performance of the methods discussed above on the EgoDexter and Dexter+Object datasets, respectively. In both cases, the method incorporating anatomical constraints performs the worst. For the EgoDexter dataset, we observe that methods utilizing weighted fingertip loss functions consistently outperform those with standard loss functions, forming a distinct cluster of higher-performing models. However, in the Dexter+Object dataset, this distinction is less pronounced.

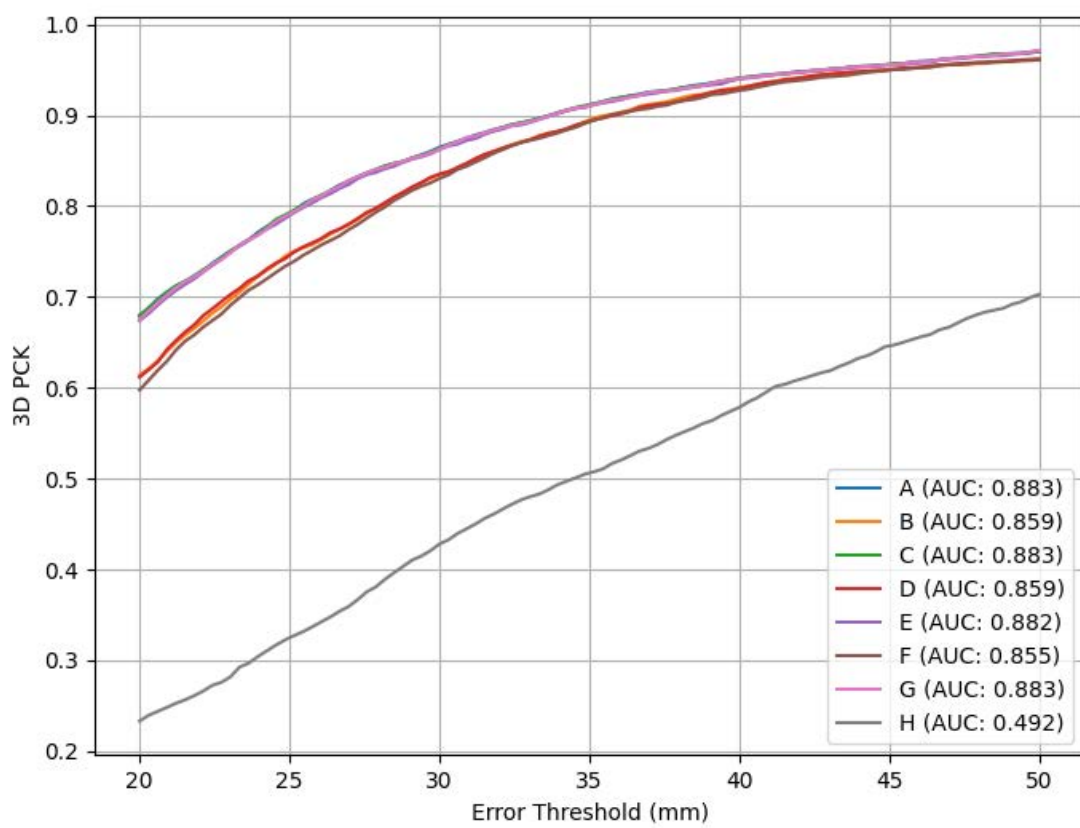


Figure 7.1: 3D PCK for EgoDexter dataset.

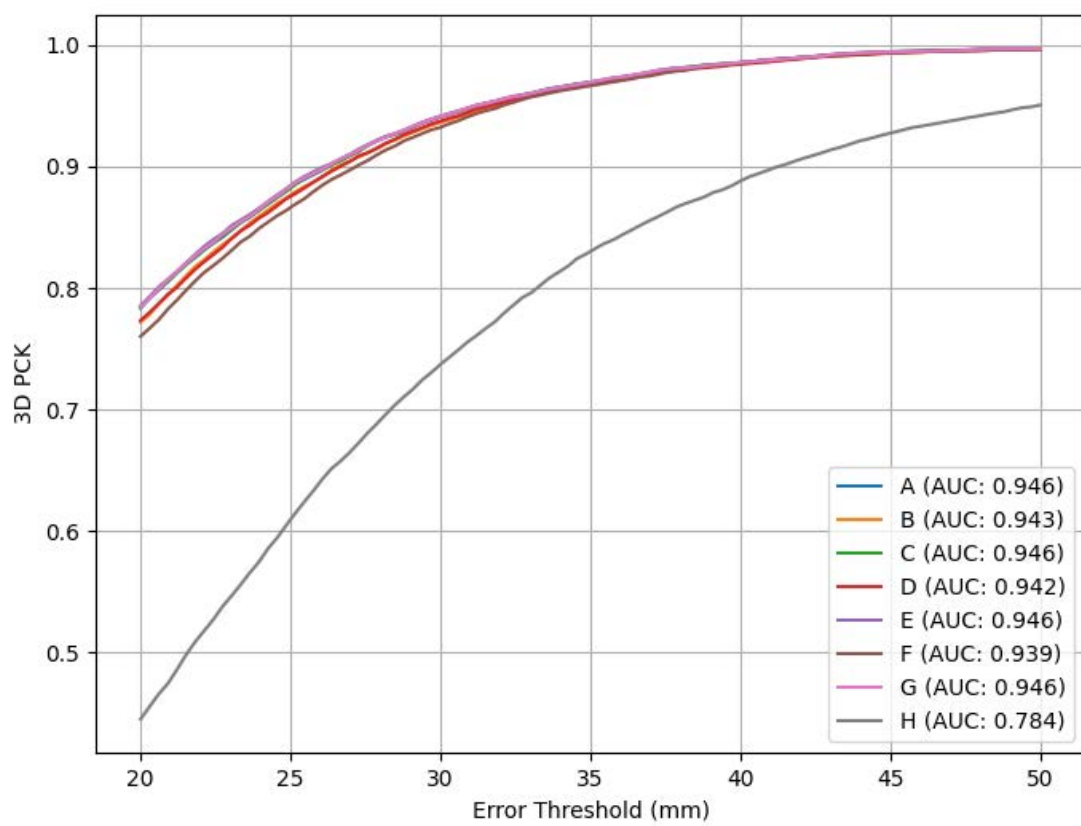


Figure 7.2: 3D PCK for Dexter+Object dataset.

7.1.5 Comparison with SotA Methods

Table 7.5: AUC of PCK ($\uparrow$) comparison with state-of-the-art methods. We use “*” to note the methods that work in real-time that are more challenging.

Method	Dexter+Object	EgoDexter
Ours	0.946	0.883
Zhou <i>et al.</i> [46]	0.948*	0.811*
Zhang <i>et al.</i> [20]	0.825	-
Baek <i>et al.</i> [17]	0.650	-
Xiang <i>et al.</i> [58]	0.912	-
Boukhayma <i>et al.</i> [4]	0.763	0.674
Iqbal <i>et al.</i> [3]	0.672	0.543
Spurr <i>et al.</i> [59]	0.511	-
Mueller <i>et al.</i> [16]	0.482*	-
Zimmermann and Brox [13]	0.573	-
Drosakis and Argyros [21]	0.764	0.563

Table 7.5 presents a comparison of our proposed method against SotA approaches, reporting the AUC of PCK on both the Dexter+Object and EgoDexter datasets. The table includes both optimization-based and learning-based methods.

Our method achieves the highest AUC on the EgoDexter dataset, outperforming all other approaches. For the Dexter+Object dataset, our method is nearly on par with the best-performing approach, with only a marginal difference of 0.002 compared to Zhou *et al.* [46].

Notably, our method not only surpasses other optimization-based methods such as Drosakis and Argyros [21] and Boukhayma *et al.* [4], but it also outperforms several learning-based methods. This is particularly significant since learning-based methods typically require substantial computational resources for training.

Furthermore, by comparing Table 7.5 with Table 7.4, we observe that not only does our best-performing configuration achieve SotA results, but even several of our alternative setups remain competitive with the top methods in the field.

Finally, it is important to emphasize that the datasets used for evaluation were not included in the training phase of any learning-based methods, ensuring a fair comparison with SotA ap-

proaches. Additionally, we strictly followed the evaluation pipeline of Zhou *et al.* [46], who reported a similar table. This guarantees a direct and unbiased comparison, further validating the superior performance of our approach.

7.2 Qualitative Evaluation

To further assess the performance of our method, we present qualitative results across various scenarios in Figures 7.3–7.8.

Figures 7.3 and 7.4 showcase examples from the EgoDexter dataset. In the simpler case (Figure 7.3), our method accurately predicts the hand keypoints. However, in the more challenging scenario (Figure 7.4), where occlusion from object interaction occurs, the method still performs reasonably well, though minor inaccuracies appear.

Figures 7.5 and 7.6 compare the effect of different loss functions on the Dexter+Object dataset. The simple MSE loss (Figure 7.5) results in less accurate predictions, particularly in fingertip locations. By contrast, using a weighted MSE loss with fingertip emphasis (Figure 7.6) improves the prediction, reinforcing the effectiveness of this approach in specific cases.

In Figure 7.7, we present a failure case caused by extreme occlusion. The EgoDexter dataset does not provide ground-truth keypoints for such cases, making evaluation difficult. Additionally, MediaPipe fails to detect a hand in this scenario, which directly impacts our method, as we rely on its initial keypoint predictions rather than ground-truth annotations from a dataset. However, in less challenging cases where MediaPipe successfully detects a hand, our optimization-based approach remains effective.

Finally, in Figure 7.8, we demonstrate that our method generalizes beyond structured datasets by estimating hand poses from an image of the Mona Lisa painting. This proves that our approach does not rely on camera parameters and can function on in-the-wild RGB images, making it applicable in diverse real-world scenarios.



Figure 7.3: Results on an image with a simple hand pose from the EgoDexter dataset. The method performs well in this scenario, accurately capturing the keypoints.

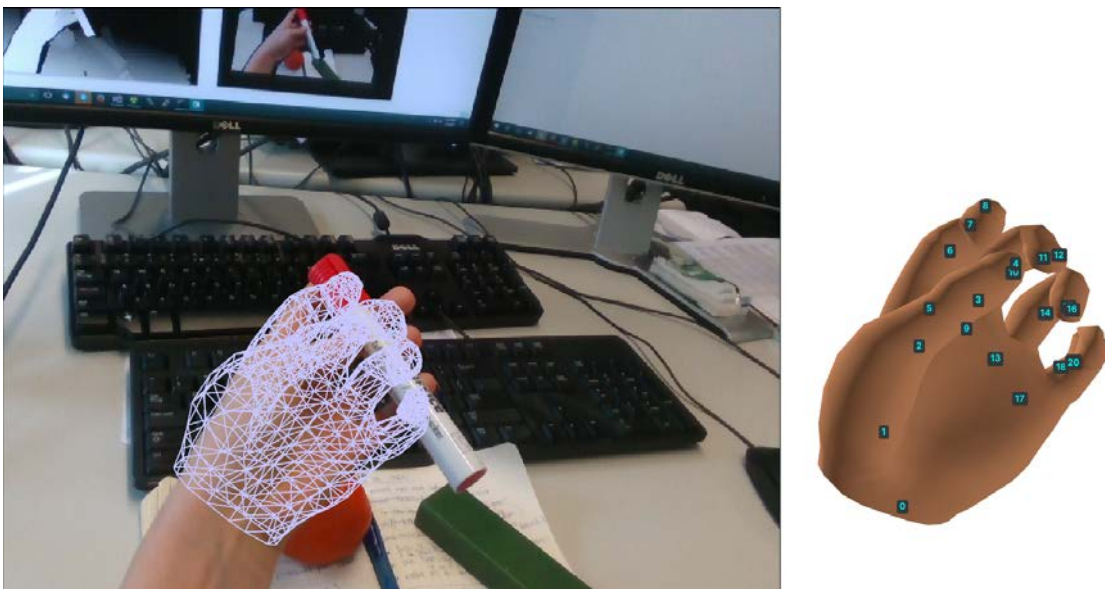


Figure 7.4: Results on an image from the EgoDexter dataset featuring a challenging hand pose with occlusion due to object interaction. Despite the occlusion, the method provides reasonable predictions.

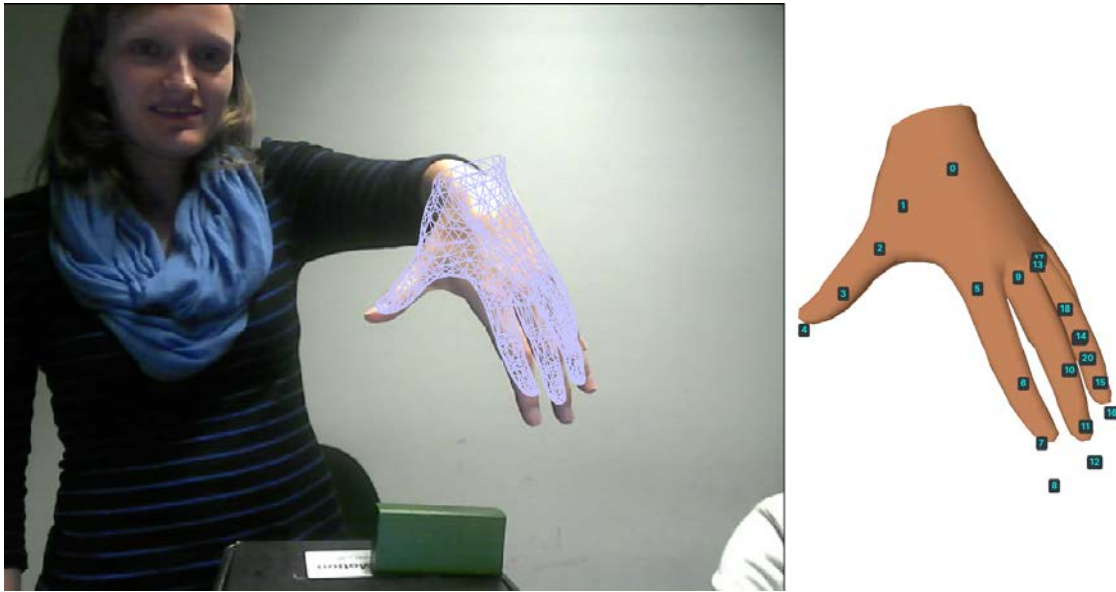


Figure 7.5: Results on an image from the Dexter+Object dataset using a simple MSE loss function. The predicted hand pose shows small inaccuracies, particularly in fingertip locations.



Figure 7.6: Results on the same Dexter+Object image as Figure 7.5, but using a weighted MSE loss function with fingertip emphasis. The hand pose estimation is significantly improved, demonstrating the advantage of this loss function in specific cases.



Figure 7.7: Example from the EgoDexter dataset where our method fails due to severe occlusion. The dataset does not provide ground truth keypoints in this particular case. Meanwhile, MediaPipe also fails to detect the hand.

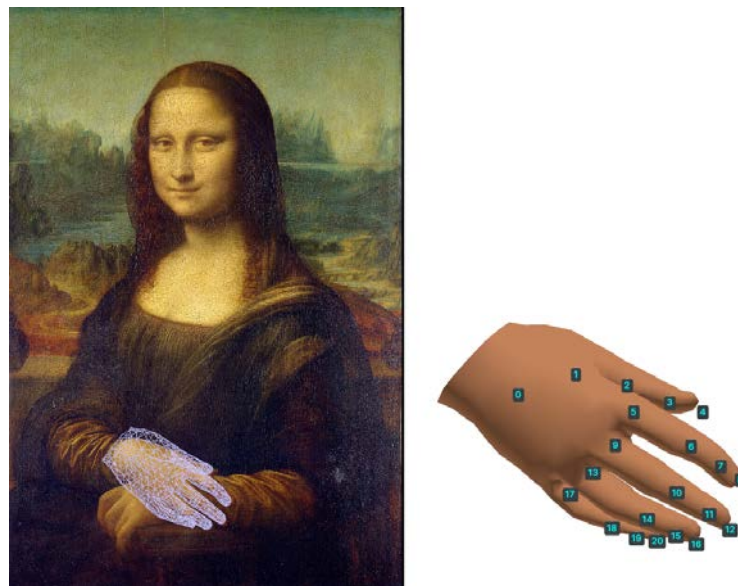


Figure 7.8: Hand pose estimation on an image of the famous Mona Lisa painting. This demonstrates that our method generalizes to standard RGB images and can work on images in the wild, independent of camera parameters.

Chapter 8

Conclusion

8.1 Summary

In this thesis, we addressed the problem of obtaining an articulated 3D hand from a single RGB image, while preserving its original pose and rotation, without requiring camera parameters or extensive training on large datasets. Recovering an accurate articulated 3D hand is a fundamental challenge in computer vision, with applications in augmented reality, virtual reality, human-computer interaction, and robotics. Existing methods often rely on deep learning models trained on large-scale annotated datasets or optimization techniques that require careful initialization and strong priors. Our work bridges these approaches by leveraging a lightweight optimization-based framework that produces 3D hands using only keypoints extracted from simple 2D images without additional depth information.

Our method utilizes MediaPipe for 2D keypoint detection and the MANO parametric hand model for 3D representation. We formulate an optimization pipeline that aligns the detected 2D keypoints with the 3D hand model, while incorporating constraints such as weighted fingertip losses and anatomical limits. Through an extensive evaluation, we demonstrated that our approach is competitive with SotA learning-based methods, despite not relying on large-scale training.

Our quantitative results on the EgoDexter and Dexter+Object datasets show that our method achieves the best performance on EgoDexter and is highly competitive on Dexter+Object, with only a 0.002 difference in AUC of PCK compared to the best-performing learning-based method. Furthermore, our approach outperforms several existing optimization-based methods, highlighting its effectiveness.

From a qualitative perspective, we demonstrated that our method works robustly on real-world images, including uncontrolled settings without known camera parameters. Additionally, we analyzed failure cases, particularly in scenarios with severe occlusion, where MediaPipe itself fails to detect keypoints. Since our method relies on these initial detections, it cannot work if the keypoints are missing. However, when MediaPipe provides reasonable estimates, our optimization produces accurate 3D hands.

By not requiring camera parameters or a dedicated training phase, our method offers greater flexibility and applicability to diverse real-world scenarios. The results indicate that optimization-based approaches can still compete with deep learning methods while being computationally efficient and adaptable to new datasets without requiring retraining.

8.2 Future Work

Several directions can be explored to extend and enhance our method. One immediate avenue for future work is evaluating our approach on additional datasets. This would provide a broader comparison with other SotA methods that did not use EgoDexter or Dexter+Object. This would also be an interesting direction, as we could assess the impact of the weighted fingertips loss function on datasets that provide all 21 hand keypoints per frame, rather than only the fingertips. This would help determine whether the weighting strategy remains beneficial when more complete keypoint annotations are available.

Extending our method to handle multiple hands in a single image is another promising area for improvement. Addressing challenges such as hand occlusions, inter-hand interactions, and keypoint associations would enhance the applicability of our approach in multi-hand scenarios.

Additionally, optimizing the computational efficiency of our method is crucial for real-world deployment. Exploring ways to accelerate the optimization process and reduce inference time could make our approach closer to real-time performance, making it viable for interactive applications.

Finally, our method could be integrated with broader tasks, such as hand-object interaction modeling or simultaneous 3D reconstruction of human bodies and hands. These extensions would further demonstrate the versatility of our approach and its potential impact in various computer vision applications.

Bibliography

- [1] Olga Chernytska. Gentle introduction to 2D hand pose estimation: Approach explained, 2021. <https://towardsdatascience.com/gentle-introduction-to-2d-hand-pose-estimation-approach-explained-4348d6d79b11> [Accessed: (11/2/2025)].
- [2] Tomas Simon, Hanbyul Joo, I. Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4645–4653, 2017.
- [3] Umar Iqbal, Pavlo Molchanov, Thomas M. Breuel, Juergen Gall, and Jan Kautz. Hand pose estimation via latent 2.5D heatmap regression. In *European Conference on Computer Vision*, 2018.
- [4] Adnane Boukhayma, Rodrigo de Bem, and Philip H. S. Torr. 3D hand shape and pose from images in the wild. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10835–10844, 2019.
- [5] Liuhao Ge, Zhou Ren, Yuncheng Li, Zehao Xue, Yingying Wang, Jianfei Cai, and Jun-song Yuan. 3D hand shape and pose estimation from a single RGB image. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10825–10834, 2019.
- [6] Guan Ming Lim, Prayook Jatesiktat, and Wei Tech Ang. MobileHand: Real-time 3D hand shape and pose estimation from color image. In *International Conference on Neural Information Processing*, 2020.
- [7] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6), November 2017.

- [8] Hand landmarks detection guide. https://developers.google.com/mediapipe/solutions/vision/hand_landmarker [Accessed: (11/2/2025)].
- [9] John Lin, Ying Wu, and T.S. Huang. Modeling the constraints of human hand motion. In *Proceedings Workshop on Human Motion*, pages 121–126, 2000.
- [10] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. MediaPipe: A framework for building perception pipelines. *ArXiv*, abs/1906.08172, 2019.
- [11] Franziska Mueller, Dushyant Mehta, Oleksandr Sotnychenko, Srinath Sridhar, Dan Casas, and Christian Theobalt. Real-time hand tracking under occlusion from an ego-centric RGB-D sensor. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2017.
- [12] Srinath Sridhar, Franziska Mueller, Michael Zollhoefer, Dan Casas, Antti Oulasvirta, and Christian Theobalt. Real-time joint tracking of a hand manipulating an object from RGB-D input. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016.
- [13] Christiane Zimmermann and Thomas Brox. Learning to estimate 3D hand pose from single RGB images. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4913–4921, 2017.
- [14] Christiane Zimmermann, Duygu Ceylan, Jimei Yang, Bryan C. Russell, Max Argus, and Thomas Brox. FreiHAND: A dataset for markerless capture of hand pose and shape from single RGB images. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 813–822, 2019.
- [15] Paschalis Panteleris, Iasonas Oikonomidis, and Antonis A. Argyros. Using a single RGB frame for real time 3D hand pose estimation in the wild. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 436–445, 2017.
- [16] Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, and Christian Theobalt. Gnerated hands for real-time 3D hand

- tracking from monocular RGB. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [17] Seungryul Baek, Kwang In Kim, and Tae-Kyun Kim. Pushing the envelope for RGB-based dense 3D hand pose estimation via neural rendering. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1067–1076, 2019.
- [18] Point cloud alignment in open3D using the iterative closest point (ICP) algorithm. <https://medium.com/@BlanchR2/point-cloud-alignment-in-open3d-using-the-iterative-closest-point-icp-algorithm-22433693aa8a> [Accessed: (11/2/2025)].
- [19] Dominik Kulon, Riza Alp Güler, Iasonas Kokkinos, Michael M. Bronstein, and Stefanos Zafeiriou. Weakly-supervised mesh-convolutional hand reconstruction in the wild. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4989–4999, 2020.
- [20] Xiong Zhang, Qiang Li, Wenbo Zhang, and Wen Zheng. End-to-end hand mesh recovery from a monocular RGB image. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2354–2364, 2019.
- [21] Drosakis Drosakis and Antonis A. Argyros. 3D hand shape and pose estimation based on 2D hand keypoints. *Proceedings of the 16th International Conference on Pervasive Technologies Related to Assistive Environments*, 2023.
- [22] Minimal-IK solver for MANO, SMPL, and SMPL-H. <https://github.com/CalciFerZh/Minimal-IK> [Accessed: (11/2/2025)].
- [23] Manotorch: framework implementing MANO in PyTorch. <https://github.com/lixiny/manotorch> [Accessed: (11/2/2025)].
- [24] Rolandos Alexandros Potamias, Stylianos Ploumpis, Stylianos Moschoglou, Vasileios Triantafyllou, and Stefanos Zafeiriou. Handy: Towards a high fidelity 3D hand shape and appearance model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4670–4680, 2023.

- [25] Neng Qian, Jiayi Wang, Franziska Mueller, Florian Bernard, Vladislav Golyanik, and Christian Theobalt. HTML: A parametric hand texture model for 3D hand reconstruction and personalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020.
- [26] Yuwei Li, Minye Wu, Yuyao Zhang, Lan Xu, and Jingyi Yu. Piano: A parametric hand bone model from magnetic resonance imaging. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 816–822, 2021.
- [27] OpenPose. <https://github.com/CMU-Perceptual-Computing-Lab/openpose> [Accessed: (11/2/2025)].
- [28] MMPose hand keypoint estimation. <https://mmpose.readthedocs.io/en/latest/demos.html#hand-keypoint-estimation> [Accessed: (11/2/2025)].
- [29] OpenCV. <https://opencv.org/> [Accessed: (11/2/2025)].
- [30] rerun.io: The multimodal data stack. <https://rerun.io/> [Accessed: (11/2/2025)].
- [31] Rerun: Time-aware multimodal data stack and visualizations. <https://github.com/rerun-io/rerun> [Accessed: (11/2/2025)].
- [32] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. The MIT Press, 2019.
- [33] A gentle introduction to the BFGS optimization algorithm. <https://machinelearningmastery.com/bfgs-optimization-in-python/> [Accessed: (11/2/2025)].
- [34] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, 2006.
- [35] Jorge Nocedal. Updating Quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.
- [36] Mean squared error (MSE), probabilitycourse.com. https://www.probabilitycourse.com/chapter9/9_1_5_mean_squared_error_MSE.php [Accessed: (9/2/2025)].

- [37] Stuart Geman and Donald E. McClure. Statistical methods for tomographic image reconstruction. *Bulletin of the International Statistical Institute*, 52(4):5–21, 1987.
- [38] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. Expressive body capture: 3D hands, face, and body from a single image. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10967–10977, 2019.
- [39] Peter J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35:492–518, 1964.
- [40] Dimitrios Tzionas, Luca Ballan, Abhilash Srikantha, Pablo Aponte, Marc Pollefeys, and Juergen Gall. Capturing hands in action using discriminative salient points and physics simulation. *International Journal of Computer Vision*, 118(2):172–193, March 2016.
- [41] scipy.optimize. minimize. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html> [Accessed: (11/2/2025)].
- [42] Rigid transformation in 3D space: Translation and rotation. <https://medium.com/@parkie0517/rigid-transformation-in-3d-space-translation-and-rotation-d701d8859ba8> [Accessed: (11/2/2025)].
- [43] numpy.linalg.svd. <https://numpy.org/doc/2.2/reference/generated/numpy.linalg.svd.html> [Accessed: (11/2/2025)].
- [44] open3D.registration.TransformationEstimationPointToPoint. https://www.open3d.org/docs/0.8.0/python_api/open3d.registration.TransformationEstimationPointToPoint.html [Accessed: (11/2/2025)].
- [45] Constantinos Voglis and Isaac E. Lagaris. A rectangular trust-region approach for unconstrained and box-constrained optimization problems. *International Conference of Computational Methods in Sciences and Engineering 2004 (ICCMSE 2004)*, 2019.

- [46] Yuxiao Zhou, Marc Habermann, Weipeng Xu, Ikhsanul Habibie, Christian Theobalt, and Feng Xu. Monocular real-time hand shape and motion capture using multi-modal data. In *Proceedings of the IEEE International Conference on Computer Vision*, 2020.
- [47] Xiong Zhang, Hongsheng Huang, Jianchao Tan, Hongmin Xu, Cheng Yang, Guozhu Peng, Lei Wang, and Ji Liu. Hand image understanding via deep multi-task learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11281–11292, 2021.
- [48] Gyeongsik Moon, Shoou-I Yu, He Wen, Takaaki Shiratori, and Kyoung Mu Lee. InterHand2.6M: A dataset and baseline for 3D interacting hand pose estimation from a single RGB image. In *European Conference on Computer Vision (ECCV)*, 2020.
- [49] Cong Peng Yangang Wang and Yebin Liu. Mask-pose cascaded CNN for 2D hand pose estimation from single color images. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(11):3258 – 3268, 2019.
- [50] Francisco Gomez-Donoso, Sergio Orts, and Miguel Cazorla. Large-scale multiview 3D hand pose dataset. *ArXiv*, abs/1707.03742, 2017.
- [51] Shicheng Chen Angela Yao Ziwei Yu, Linlin Yang. Local and global point cloud reconstruction for 3D hand pose estimation. In *British Machine Vision Conference (BMVC)*, 2021.
- [52] Samarth Brahmabhatt, Chengcheng Tang, Christopher D. Twigg, Charles C. Kemp, and James Hays. ContactPose: A dataset of grasps with object contact and hand pose. In *European Conference on Computer Vision (ECCV)*, 2020.
- [53] Fanqing Lin, Connor Wilhelm, and Tony Martinez. Two-hand global 3D pose estimation using monocular RGB. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2373–2381, 2021.
- [54] Yana Hasson, Gül Varol, Dimitrios Tzionas, Igor Kalevatykh, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning joint reconstruction of hands and manipulated objects. In *CVPR*, 2019.

- [55] Jiawei Zhang, Jianbo Jiao, Mingliang Chen, Liangqiong Qu, Xiaobin Xu, and Qingxiong Yang. 3D hand pose tracking and estimation using stereo matching. *ArXiv*, abs/1610.07214, 2016.
- [56] Grégory Rogez, James Steven Supani, Maryam Khademi, José M. M. Montiel, and Deva Ramanan. 3D hand pose detection in egocentric RGB-D images. *ECCV Workshops*, abs/1412.0065, 2014.
- [57] Srinath Sridhar, Antti Oulasvirta, and Christian Theobalt. Interactive markerless articulated hand motion tracking using RGB and depth data. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2013.
- [58] Donglai Xiang, Hanbyul Joo, and Yaser Sheikh. Monocular total capture: Posing face, body, and hands in the wild. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10957–10966, 2018.
- [59] Adrian Spurr, Jie Song, Seonwook Park, and Otmar Hilliges. Cross-modal deep variational hand pose estimation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 89–98, 2018.

Appendix

In this appendix we present our implementation of two loss functions and the rigid transform method function

A.1 Custom python loss functions

The Geman-McClure loss and the Huber loss functions are described in chapter 5.

```
# Geman-McClure loss calculation function
def GMLossCalc(residual, rho=1):
#residual is the difference x_estimated-x_real. Default rho=1
    gm_loss = torch.div(rho ** 2 * residual ** 2,
        residual ** 2 + rho ** 2)
    return gm_loss.mean()
# .mean() to ensure its a scalar for loss.backward()

# Huber loss calculation function
def huber_loss(pred, target, delta=1.0):
    error = pred - target
    abs_error = torch.abs(error)
    quadratic = 0.5 * error ** 2
    linear = delta * (abs_error - 0.5 * delta)
    return torch.where(abs_error < delta,
        quadratic, linear).mean()
```

A.2 Rigid transformation

We implemented the rigid transformation method (see also chapter 6) in Python using NumPy, as shown in the following function:

```
def rigidTrans(source, target):
    """
    Registers two point clouds by rigid transformation using
    Singular Value Decomposition (SVD).
    target_x = target_T_source * source_x
    """
    assert len(source) == len(target)
    #Source and target must have the same number of points

    # Compute centroids of the source and target point clouds
    source_centroid = np.mean(source, axis=0)
    target_centroid = np.mean(target, axis=0)

    # Center the point clouds around their centroids
    source_centered = source - source_centroid
    target_centered = target - target_centroid

    # Compute the covariance matrix
    H = np.dot(source_centered.T, target_centered)

    # Compute the Singular Value Decomposition (SVD)
    #of the covariance matrix
    U, _, Vt = np.linalg.svd(H)

    # Compute the optimal rotation matrix
    R = np.dot(Vt.T, U.T)

    # Handle the case where the determinant of R is -1
    #(reflection)
```

```

if np.linalg.det(R) < 0:
    Vt[-1, :] *= -1
    R = np.dot(Vt.T, U.T)

    # Compute the translation vector
    t = target_centroid - np.dot(source_centroid, R.T)

    # Construct the 4x4 transformation matrix
    transformation_matrix = np.eye(4)
    transformation_matrix[:3, :3] = R
    transformation_matrix[:3, 3] = t

    return transformation_matrix

```

Here is also the function used to reverse this transformation that is used in the final step of our optimization pipeline:

```

# Reverse rigidTrans transformation
def reverse_rigid_transformation(keypoints,
    transformation_matrix):
    """
    Reverses a rigid transformation applied to keypoints.

    Args:
    - keypoints (np.ndarray): Nx3 array of transformed
      3D keypoints.
    - transformation_matrix (np.ndarray): 4x4 rigid
      transformation matrix.

    Returns:
    - reversed_keypoints (np.ndarray): Nx3 array of
      keypoints in the original coordinate space.
    """

```

```
# Extract rotation and translation  
#from the transformation matrix  
R = transformation_matrix[:3, :3] #3x3 rotation matrix  
t = transformation_matrix[:3, 3] #3x1 translation vector  
  
# Compute the inverse transformation  
R_inv = R.T # Transpose of rotation matrix  
t_inv = -np.dot(R_inv, t) # Inverse translation vector  
  
# Apply the inverse transformation to keypoints  
keypoints_transformed = np.dot(keypoints - t, R_inv.T)  
return keypoints_transformed
```