BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

DEPTH-BASED DETERMINATION OF A 3D HAND POSITION

STANOVENÍ 3D POZICE RUKY V PROSTORU Z HLOUBKOVÉHO OBRAZU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **LADISLAV ONDRIS**
AUTOR PRÁCE

SUPERVISOR **Prof. Ing. MARTIN DRAHANSKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2020/2021

Bachelor's Thesis Specification

Student: **Ondris Ladislav**
Programme: Information Technology
Title: **Depth-Based Determination of a 3D Hand Position**
Category: Image Processing

Assignment:

1. Study the literature on hand detection and determining its position from depth images.
2. Design a suitable algorithm for detecting the hand, determining its position, and accepting the gesture (e.g., outstretched and fingers far apart).
3. Implement the proposed algorithm from point 2.
4. Acquire a database of at least 100 records. Use this database to test your solution from point 3.
5. Summarize the achieved results and discuss possible extensions or improvements.

Recommended literature:

- SUPANČIČ, James Steven, et al. Depth-based hand pose estimation: methods, data, and challenges. *International Journal of Computer Vision*, 2018, 126.11: 1180-1198.
- EROL, Ali, et al. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 2007, 108.1-2: 52-73.
- BARSOUM, Emad. Articulated hand pose estimation review. *arXiv preprint arXiv:1604.06195*, 2016.

Requirements for the first semester:

- Items and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 12, 2021
Approval date: November 11, 2020

Abstract

This work aims to offer a real-time, depth-based gesture recognition system using a hand's skeletal information. The Tiny YOLOv3 neural network detects the hand in the depth image. The detected hand is rid of the background and used by the JGR-P2O neural network, which estimates the hand's skeleton represented by 21 key points. Furthermore, a novel technique for gesture recognition from hand key points that compares the input skeleton with user-defined gestures has been proposed. A dataset consisting of four thousand images was captured to evaluate the system.

Abstrakt

Cílem této práce je určení kostry ruky z hloubkového obrazu a jeho následné využití k rozpoznání statického gesta. Na vstupu je hloubkový obrázek, ve kterém je nejprve detekována ruka pomocí neuronové sítě Tiny YOLOv3. Následně je obrázek zbaven pozadí a z takto předzpracovaného obrázku je určena kostra ruky v podobě 21 klíčových bodů neuronovou sítí JGR-P2O. K rozpoznání gesta z klíčových bodů ruky byla navržena technika, která porovná kostru na vstupu s uživatelem definovanými gesty. Funkcionalita systému byla otestována na vytvořeném datasetu s více než čtyřmi tisíci obrázky.

Keywords

image processing, object detection, hand pose estimation, gesture recognition, depth image, depth-based, real-time, YOLOv3, JGR-P2O, convolutional neural network, deep learning, key points, skeleton, hand

Klíčová slova

zpracování obrazu, detekce objektů, odhad pozice ruky, rozpoznání gesta, hloubkový obrázek, reálný čas, YOLOv3, JGR-P2O, konvoluční neuronová síť, hluboké učení, klíčové body, kostra, ruka

Reference

ONDRIS, Ladislav. *Depth-Based Determination of a 3D Hand Position.* Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Ing. Martin Drahanský, Ph.D.

Rozšířený abstrakt

Cílem této práce je především určení kostry ruky v hloubkovém obraze, a dále potom využití této informace k rozpoznání statického gesta v reálném čase, přičemž toto gesto si definuje uživatel sám. Na vstupu systému je hloubkový obrázek, v němž nejprve detekuje ruku. Existuje několik přístupů pro detekci ruk v obrázcích. Například detekce ruky na základě barvy pleti, identifikace tělesné teploty či extrakce nejbližšího objektu před kamerou. Tato práce využívá hloubkové obrázky, jež obsahují informace o vzdálenosti objektů. K samotné detekci pak používá konvoluční neuronovou síť Tiny YOLOv3, která je trénovaná na datasetu HandSeg. Byla vybrána především pro svoji robustnost oproti alternativním způsobům detekce. Úspěšnost lokalizace objektů této metody dosahuje 86.35 % IoU na testovací části datasetu. Vyhodnocení na reálných obrázcích prokázalo použitelnost tohoto modelu pro další části systému.

Detekovaná ruka je následně vystřižena z obrázku a je zbavena pozadí, které by mohlo ovlivňovat další části systému. Toto předzpracování je složeno z několika kroků, mezi které patří extrakce ohraničující krychle, aplikace Otsovy prahové metody, a nakonec ponechání pouze největšího objektu v obrázku. Takto předzpracovaný obrázek je potom vstupem do další části systému, kterou je určení pózy, která najde pozici 21 klíčových bodů kostry ruky. Díky pokroku v neuronových sítích, a to především konvolučních, se odborná komunita začala aktivně zabývat touto problematikou. Každým rokem se úspěšnost navržených modelů zvyšuje, z čehož vyvstává příležitost použití informace o kostře k rozpoznání gesta, které dříve muselo spoléhat na jiné metody, například založené na kontuře. Ukazuje se, že úspěšnost modelu pro určení pózy je silně ovlivněna datasetem, na kterém je trénován, což se projevuje především v reálných podmínkách, kdy model vrací nepřesné výsledky na neviděné pózy. Toto chování se ukazálo také na vlastním datasetu. Jako model pro určení kostry byla vybrána konvoluční neuronová síť JGR-P2O, jež byla trénována na dvou různých datasetech. Její úspěšnost byla vyhodnocena metrikou počítající průměrný rozdíl mezi předpovězenou pozicí a skutečnou pozicí klíčových bodů. Na datasetu MSRA15 dosahuje tento rozdíl 14.7 milimetrů a na datasetu BigHand přibližně 24.9 milimetrů, což je dáno především jeho výraznou složitostí v porovnání s datasetem MSRA15.

Navržené řešení pro rozpoznání gesta zakladající na znalosti kostry ruky poskytuje výhodu v podobě flexibility stanovení rozpoznávaného gesta a stanovení maximálního natočení ruky vůči cílovému gestu. Naproti tomu nevýhodou tohoto řešení je nutnost přesného určení kostry, což je značně obtížná úloha. Natočení ruky ke kameře je dáno proložením klíčových bodů dlaně rovinou. Póza ruky je reprezentována metrikou spočtenou na základě vzájemných vzdáleností mezi každou dvojicí klíčových bodů ruky. Tyto vzdálenosti jsou porovnány se vzdálenostmi spočtenými nad pózami cílového gesta. Pokud toto porovnání dosahuje hodnoty nepřekračující určitý práh, pak je póza považována za cílové gesto. Toto řešení bylo vyhodnoceno na anotacích datasetu MSRA, který rozlišuje mezi 17 gesty. Tento systém je schopný dosáhnout více než 95% přesnosti při relativně malém množství vzorových obrázků.

Výsledný systém byl otestován na vlastním datasetu. Navíc byla testována i schopnost rozpoznání v reálném čase. Vyhodnocením bylo zjištěno, že celková úspěšnost systému je degradovaná nedostatečnou přesností modelu pro určení kostry. Systém funguje relativně dobře pro jedno ze základních gest, nicméně vyžaduje přesnější stanovení pózy pro použití v širším spektru aplikací vyžadujících různá gesta.

Depth-Based Determination of a 3D Hand Position

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Prof. Ing. Martin Drahanský Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Ladislav Ondris
May 10, 2021

Acknowledgements

I want to thank my supervisor, Prof. Ing. Martin Drahanský Ph.D., for his professional guidance and consultation. My thanks also go to my friends who supported me. I would especially like to thank Alexander Polok for his advice and countless discussions.

I am also grateful to MetaCentrum for providing me with computational resources. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Contents

1 Introduction **2**

2 State-of-the-art **3**
2.1 Object detection 3
2.2 Depth-based hand pose estimation 6
2.3 Hand gesture recognition 11

3 Used architectures **14**
3.1 You Only Look Once 14
3.2 Joint Graph Reasoning based Pixel-to-Offset Prediction Network 16

4 Datasets **23**
4.1 Challenges in hand position determination 23
4.2 Hand segmentation datasets 24
4.3 Hand pose estimation datasets 25
4.4 A custom dataset 29

5 Design of gesture acceptance system **31**
5.1 System requirements 31
5.2 Proposed design 32

6 Implementation and training **42**
6.1 Toolset 42
6.2 Recognition system 43
6.3 Training 45

7 Evaluation and testing **50**
7.1 Evaluation metrics for binary classifiers 50
7.2 Hand detection 51
7.3 Hand pose estimation 53
7.4 Gesture recognition on MSRA15 gesture dataset 56
7.5 System evaluation 57

8 Conclusion **61**

Bibliography **63**

Chapter 1

Introduction

Gesture recognition is one of many challenging tasks in image processing that have found applications in various industries. It is vital in the automotive industry, home automation, and healthcare, and pivotal to human-computer interaction such as augmented reality.

Interpreting a gesture can be done in many ways. However, one commonality is that most techniques rely on hands' key points represented in a 3D coordinate system. The key points can be extracted using vision-based hand pose estimation, which is currently a highly active research area. The vision-based techniques use color or depth cameras for capturing images from which hands are extracted using object detection techniques.

The boom in machine learning and, specifically, deep learning opened the door to many complex tasks that would be very difficult to solve with algorithmic approaches, especially tasks in image processing such as gesture recognition and hand pose estimation. This work aims to utilize modern techniques for object detection and hand pose estimation to perform gesture recognition in real-time. Most works tackling gesture recognition categorize the hand's pose into a predefined set of gestures. This work aims to develop a system that gives the user the possibility to set the desired gesture as they need and provides them with feedback on the gesture. Such features necessitate the creation of a novel gesture recognition system, which could be adopted in many applications, such as hand scanning.

The latest methods for addressing the above-mentioned tasks are introduced in Chapter 2, which discusses their basic characteristics. Some of them are employed in the final system for gesture recognition. These are described in greater detail in Chapter 3. Chapter 4 presents an overview of the available datasets that may be employed in training relevant models, and describes those used in this work more thoroughly. The system's design is presented in Chapter 5, which focuses on connecting all three tasks. The training and implementation of the system are discussed in Chapter 6. Each task was evaluated separately, and the complete system was tested in real conditions, with results presented in Chapter 7. Chapter 8 summarizes the achieved results from the previous chapter and discusses possible improvements.

Chapter 2

State-of-the-art

This chapter introduces three tasks related to determining the hand's position and gestures. It also provides a short overview of the state-of-the-art methods for solving them. The first task is object detection for locating objects in images. It is followed by an introduction to hand pose estimation for determining the position of the hand's skeleton. The chapter is concluded with a brief introduction to gesture recognition with a few examples of gesture recognition techniques.

2.1 Object detection

Object detection is the process of estimating objects' locations in images. It is composed of two parts: object localization and object classification. Object localization tells where objects are located in the image, whereas object classification assigns classes to these objects. Many applications require object detection. One of many examples is a self-driving car which needs to locate surrounding objects and classify them. The object detector in this work performs binary classification to recognize hands in depth images. The two classes are hands and background.

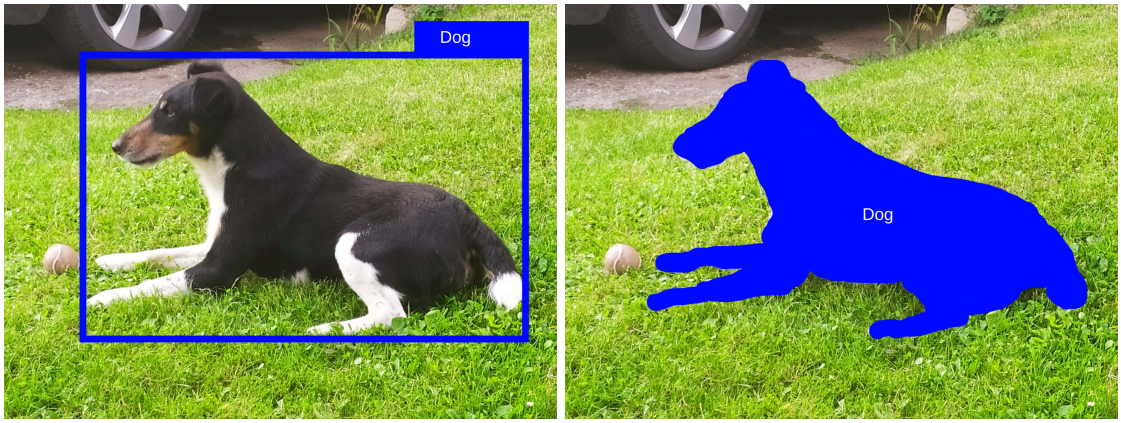
An alternative technique to object detection exists—image segmentation [9]. It assigns classes to each pixel in a way that creates homogeneous regions of the same object category. Salient detectors often adopt this technique. Figure 2.1 shows the difference between object detection and image segmentation. This thesis uses an object detector, and, therefore, any additional details on image segmentation are not presented.

Zhong et al. [70] divide object detection by application into two primary domains—salient objection detection and generic object detection.

2.1.1 Salient object detection

Salient object detection [5] is interpreted as a process consisting of two stages: detecting the most dominant (salient) object in the image and segmenting the object's accurate region. However, most methods do not explicitly distinguish between the two stages.

The first stage usually locates one but sometimes more dominant objects. The second stage is similar to traditional image segmentation, with the difference that the most salient object determines the score.



(a) (b)

Figure 2.1: Object detection assigns classes to each detected object **(a)**, whereas segmentation assigns a class label to each pixel in the image **(b)**.

2.1.2 Generic object detection

Generic object detection determines whether an image contains objects of predefined categories and, if so, returns the locations of these objects together with the category labels.

Generic object detection has shown tremendous progress in the last two decades. Many architectures have proved themselves to be excellent detectors. Figure 2.2 shows essential detectors developed in recent years. Every generic object detector can be categorized either as a one-stage or two-stage.

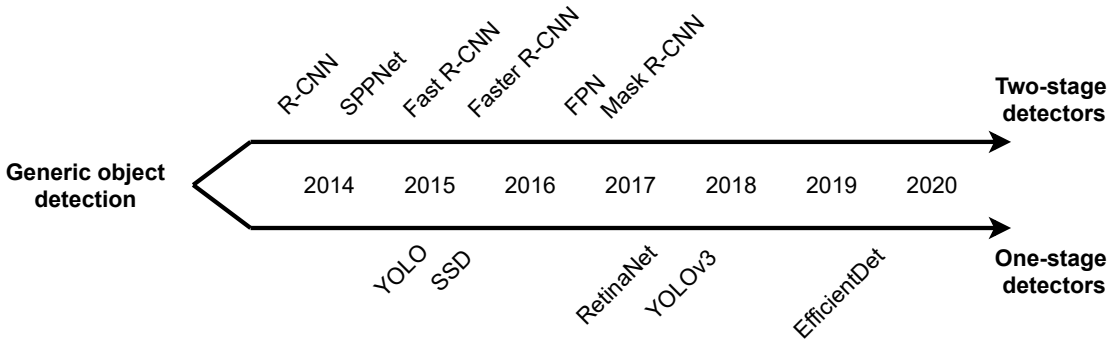


Figure 2.2: Generic object detectors are often divided into One-stage and Two-stage detectors. [74][33] The latter has a region proposal stage that filters out most negative locations and a second stage that classifies the proposals. The former does both tasks in a single pass.

Two-stage detectors

Two-stage detectors share the characteristic of comprising of two stages or sometimes even more. One is a region proposal stage, and the second one is a classification of these proposals.

The leading representative of this category is the R-CNN family, which stands for Region-Based Convolutional Neural Network. This family includes R-CNN [19], Fast R-CNN [18], Faster R-CNN [48], and Mask R-CNN [24]. The R-CNN proposes three stages: region proposal, feature extraction, and classification. The other R-CNN models further improve this architecture in terms of speed and accuracy.

SSPNet [25] stands for Spatial Pyramid Pooling in Deep Convolutional Networks. It equips the network with a new pooling strategy—spatial pyramid pooling—to remove the requirement of a fixed-sized image.

The FPN [32] introduced a pyramidal hierarchy of deep convolutional networks to construct feature pyramids to tackle multiscale object detection.

One-stage detectors

One-stage detectors have shown great potential in recent years. One of their advantages is their speed as it performs both stages together.

YOLO is a popular family of fast one-stage detectors. This family includes YOLO [46], YOLOv3 [47], YOLOv4 [4], and others. Their main advantage is their inference speed, which makes them suitable for real-time applications. Section 3.1 provides a more detailed description.

SSD [34] (Single Shot Detector) uses a single deep convolutional neural network. It employs multiscale feature maps to detect objects at multiple scales and associates default bounding boxes for each cell at a fixed position relative to that cell.

RetinaNet [33] is a dense detector with a new loss function, focal loss, designed to solve the background-foreground class imbalance. The focal loss down-weights well-classified classes and focuses on complex examples instead, preventing the background or other easy negatives from overwhelming the detector during training.

EfficientDet [53] is an architecture with several new optimizations to improve efficiency. One of the significant optimizations is a weighted bidirectional feature pyramid network (BiFPN).

2.1.3 Approaches to hand detection

Hand pose estimation and gesture recognition often require hand detection to extract a subregion from the original image containing the hand, which increases the estimator's performance by providing them with a stable, preprocessed input.

The scientific community has come up with many approaches to hand detection. However, each of them may be suitable for a different environment. This section is based on the Articulated Pose Estimation Review [3], in which Barsoum provided an overview of those techniques.

Skin color-based detection

Some methods use skin color to infer hand location. It suffers from many problems, such as it is not invariant to illumination and assumes that no other object in the scene has the same color as the skin. Furthermore, human skin color varies between populations, and, for this reason, it can be problematic.

Temperature-based detection

The idea is that the body temperature is constant. The hand can be easily detected using thermal vision and a threshold. However, in some cases, the body temperature may be off due to sickness and other reasons. This approach presumes that no other object with the same temperature is present in the scene.

Marker-based detection

The hand can be detected by wearing a colored glove or coloring the hand itself. However, it is unnatural to wear special gloves in everyday environments and can only be employed for exceptional cases. It is worth noting that this technique is often used to generate depth image datasets.

Motion-based detection

Given a sequence of frames, the motion-based detector locates regions of objects that change their position relative to their surroundings. It is suitable for environments with stationary backgrounds but may be faulty in crowded areas. Motion-based detection is related to object tracking.

Detection of the nearest object

Depth-based approaches assume that the hand is the nearest object to the camera. Although the method is relatively simple to implement, it is applicable only under certain circumstances since closer objects could lie in the scene, e.g., a human body, a desk, or another pad on which the camera rests. However, many authors of hand pose estimators use this method to extract hands from images. This approach may be appropriate as long as it is not restrictive for the target application.

Color-based and depth-based detection

Machine learning methods can be utilized for hand detection from color and depth images. These methods include generic object detectors from Section 2.1.2, e.g., YOLO, SSD, or image segmentation methods that predict a class label for each pixel. One such example of a segmentation model is a Random Decision Forest [50], which utilizes depth information. Their advantage is that there are no constraints on the environment or that the hand is not required to be the closest object. Furthermore, they are robust and accurate. On the other hand, they are generally more computationally intensive.

2.2 Depth-based hand pose estimation

Depth-based hand pose estimation determines the hand joints' locations in 3D space from depth images. This challenging task is pivotal to many applications, such as human-computer interaction, sign language recognition, and driver analysis. It is also helpful when the user cannot touch the screen, such as a medical doctor during an operation or controlling television. [28][3]

Figure 2.3 illustrates a skeletal hand model that consists of 21 joints—a wrist and five fingers. Each finger is represented by four joints—a metacarpophalangeal joint (MCP),

a proximal interphalangeal joint (PIP), a distal interphalangeal (DIP), and a fingertip. Most hand pose estimation models approximate this hand model.

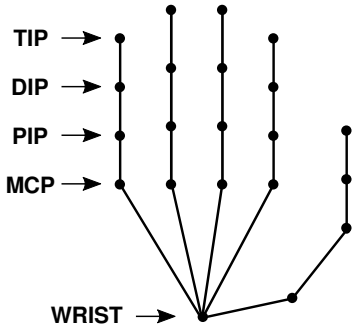


Figure 2.3: The figure shows a skeletal hand model that is used in hand pose estimation most often. The model consists of 21 joints—a wrist, MCP, DIP, PIP joints, and fingertips. The figure was adapted from the BigHand paper [66].

2.2.1 Evaluation metrics for 3D pose estimation

Evaluation metrics need to be introduced to understand the performance of models and their comparison. The commonly used metrics in hand pose estimation include the mean joint error and the proportion of joints within distance.

Mean joint error

Mean joint error, abbreviated as MJE, is defined as the average Euclidean distance between the predicted and ground truth joints in a 3D coordinate system. [43] The metric is calculated between two hands' positions:

$$\text{MJE} = \frac{1}{N}\sum_{j=1}^{N} d(p_j, q_j), \tag{2.1}$$

where N denotes the number of joints, and $d(p_j, q_j)$ is a Euclidean distance between the locations of joints p_j and q_j. It is the primary metric that all papers use to evaluate their models and compare them with other state-of-the-art methods. Such a comparison is provided in Table 2.1.

The proportion of joints within distance

This metric is defined as the proportion of frames with all predicted joints within a given maximum Euclidean distance from the ground truth annotations. This metric is challenging because a single displaced joint can decline the whole pose. [43] [11]

2.2.2 Coordinate mapping

Certain situations necessitate a conversion between pixel coordinates and world coordinates. For example, some hand pose estimation methods take volumetric space as its input. In such a case, the depth image is converted into a 3D voxel grid. [36] Other works, such as JGR-P2O [13], predict a combination of pixel coordinates and a depth value. Even though

the pixel coordinates representation has certain advantages, the mean joint error metric requires world coordinates.

The conversion is based on a pinhole camera model, which describes the perspective projection of a 3D space onto a 2D projection plane, as shown in Fig. 2.4. This type of projection does not preserve the parallelism of the edges, and the size of the object displayed on the projection plane is inversely proportional to the object's distance to the camera. The farther away the object is from the projection plane, the smaller its image. [23]

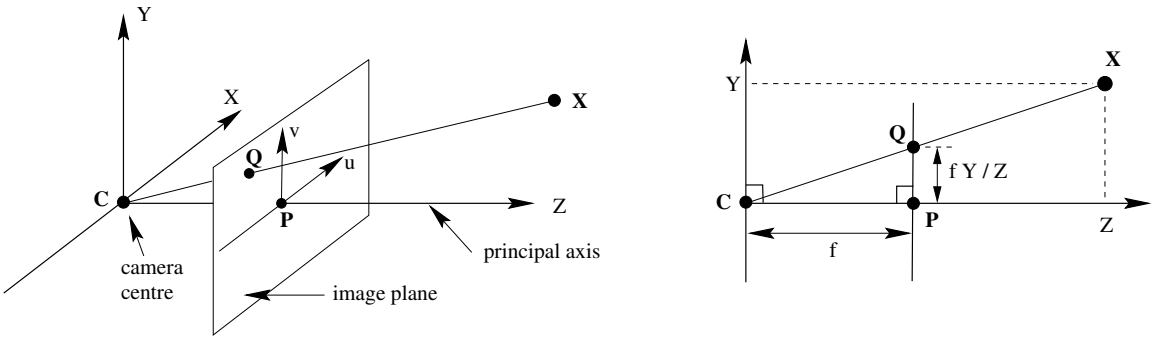


Figure 2.4: The pinhole camera model. The projection of point $\mathbf{X} = (x, y, z)$ onto the image plane is point $\mathbf{Q} = (u, v)$. Adapted from Multiple view Geometry in Computer Vision [23].

Camera aperture is located at the origin of the coordinate system $\mathbf{C}$ and is viewing in the direction of the Z axis called the principal axis. The image plane is orthogonal to the principal axis and is referred to as the principal plane or image plane. The intersection of the principal axis and the image plane is the principal point $\mathbf{P} = (p_u, p_v)$. The projection of point $\mathbf{X} = (x, y, z)$ onto the principal plane is the point $\mathbf{Q} = (u, v)$, which is the intersection of the principal plane with the line connecting camera center $\mathbf{C}$ and the point $\mathbf{X}$. The distance between the camera and the principal point is the focal length f.

Then the projection can be expressed by Equation (2.2).

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{z} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.2}$$

Equation (2.2) assumed that the origin of coordinates in the image plane is at the principal point. Often, that is not the case. Therefore, the mapping is given by Equation (2.3).

$$\begin{bmatrix} f & 0 & p_u \\ 0 & f & p_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \frac{1}{z} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.3}$$

2.2.3 HANDS17 and HANDS19 challenges

The scientific community has shown great interest in this field during the past decade. With the availability of depth cameras, new datasets could emerge with a large variability of viewpoints, hand articulation, and clutter. The viewpoint is interpreted as hand position (global orientation) and the articulation as joint angles. However, a wide variation of joint angles does not necessarily imply a large variability of hand poses.

Two public competitions occurred in 2017 and 2019, called HANDS17[1] and HANDS19[2] respectively. These competitions aimed to evaluate the present state of 3D hand pose estimation.

Each of these competitions comprised of three tasks. The HANDS17 included single frame 3D hand pose estimation, 3D hand pose tracking, and hand-object interaction. Single pose estimation is performed on individual images. Given an image, the system should estimate the joints' locations. Hand pose tracking is performed on a sequence of images. The system receives the first frame's annotations and sequentially estimates the poses on the rest of the images in the sequence. In the hand-object interaction task, the system should be able to predict the joints' locations despite objects occluding the hand. [65]

The best performing method for single pose estimation in this challenge was V2V-PoseNet [40].

The HANDS19 challenge includes single frame pose estimation and two more tasks tackling the hand-object interaction problem—with depth and color images. [2] Figure 2.5 shows the results of different methods on the second task of this competition.

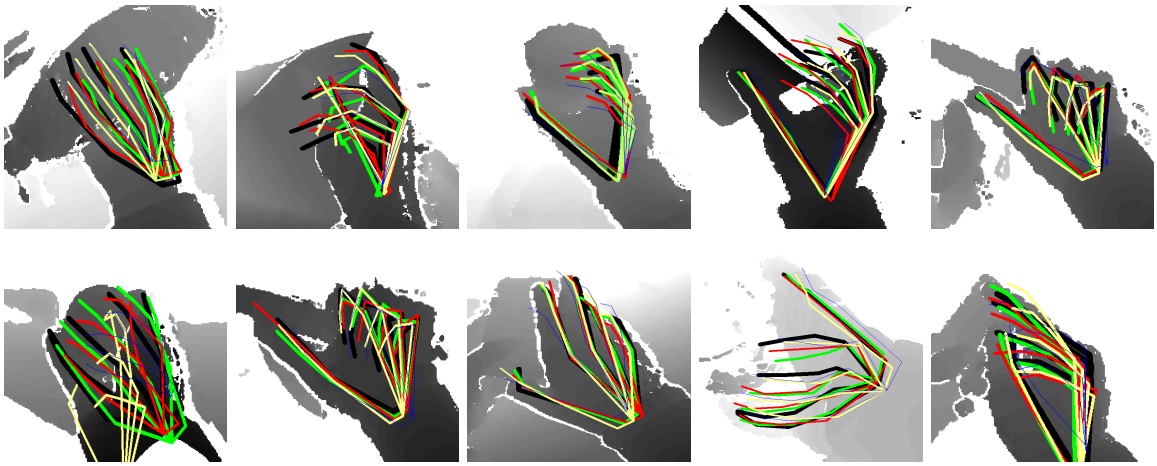


Figure 2.5: Visualization of hand pose estimation with objects from depth images, which was the second task in the HANDS19 competition. It shows estimations of NTIS, A2J, Crazy-Hand, BT, and the **ground-truth** annotation. The figure was adopted from HANDS19 summarising paper [2].

Suspancic et al. [28] provide a review of different methods and show that deep models are well-suited for hand pose estimation. Because of their well-designed networks' structures, they can solve complex problems and outperform the traditional methods in many computer vision tasks. These models benefit from the numerous fully annotated datasets which were recently created.

The authors of the JGR-P2O paper [13] provided a more recent review of the state-of-the-art methods. They can be divided into direct regression-based methods and detection-based methods. Table 2.1 indicates that the detection-based methods are superior to the regression-based ones.

[1] http://icvl.ee.ic.ac.uk/hands17/challenge/

[2] https://sites.google.com/view/hands2019/challenge

Method	Mean error (mm)			Type	Params	Speed (fps)
	ICVL	NYU	MSRA			
DeepModel [72]	11.56	17.04	-	R	-	-
DeepPrior [43]	10.40	19.73	-	R	-	-
DeepPrior++ [42]	8.10	12.24	9.50	R	-	30.0
REN-4x6x6 [20]	7.63	13.39	-	R	-	-
REN-9x6x6 [20]	7.31	12.69	9.70	R	-	-
Pose-REN [7]	6.79	11.81	8.65	R	-	-
3DCNN [16]	-	14.1	9.60	R	104.9M	215
HandPointNet [15]	6.94	10.54	8.50	R	2.58M	48.0
SHPR-Net [8]	7.22	10.78	7.76	R	-	-
CrossInfoNet [11]	6.73	10.08	7.86	R	23.8M	124.5
DenseReg [58]	7.30	10.2	7.20	D	5.8M	27.8
Point-to-Point [17]	6.30	9.10	7.70	D	4.3M	41.8
V2V-PoseNet [40]	6.28	8.42	7.59	D	457.5M	3.5
Point-to-Pose Voting [31]	-	8.99	-	D	-	80.0
A2J [62]	6.46	8.61	-	D	44.7M	105.1
JGR-P2O [13]	6.02	8.29	7.55	D	1.4M	111.2

Table 2.1: The table shows an overview of state-of-the-art methods for hand pose estimation from the JGR-P2O paper [13]. They are compared on three datasets: ICVL, NYU, and MSRA. Further details on these datasets are given in Section 4.3.

2.2.4 Detection-based methods

As shown in Table 2.1, the best performing methods are detection-based. They produce volumetric heat maps or 3D heat maps with offset vector fields for each joint. Because they take this indirect approach of determining the 3D coordinates, they can learn better feature representations. Often, this results in a lousy trade-off between accuracy and efficiency. [13]

DenseReg leverages from both 2D and 3D properties of a depth map. They decompose the pose representations into 2D heat-maps, 3D heat-maps, and unit 3D directional vector fields. [58]

V2V-PoseNet is designed to avoid the highly nonlinear mapping of directly regressing the 3D coordinates from 2D images. Instead, they propose a voxel-to-voxel prediction that uses a 3D voxelized grid and estimates the per-voxel likelihood for each joint. [40]

A2J introduces a novel anchor-based approach called an Anchor-to-joint regression network. Anchor points are densely set up on the depth image to capture global-local spatial context information. These anchor points predict the positions of joints using weighted aggregation. Each joint has different informative anchor points. [62]

The authors of the **JGR-P2O** network mitigated the problematic trade-off between accuracy and efficiency by analyzing state-of-the-art methods and proposed a solution that does not suffer from enormous computations while still preserving high efficiency. More details on the architecture of this network are provided in Section 3.2.

2.2.5 Regression-based methods

The regression-based methods map depth images to 3D hand pose representations directly, which is a highly nonlinear mapping causing slightly worse performance, in contrast to detection-based methods. [13]

DeepPrior introduced a constrained prior hand model that remarkably improves the joint localization accuracy. The prior is implemented by introducing a bottleneck layer in the network's architecture that produces a low-dimensional embedding. Above this layer is placed another hidden layer for the reconstruction of the hand pose space from the embedding. They initialize the weights of this layer with the major components of Principal Component Analysis. They also include a second stage that refines each joint's location from the first stage using a separate CNN network. They apply pooling with filters of different sizes. The smaller filters allow more precise predictions, while the larger ones provide contextual information. [43]

DeepPrior++ is an improved version of DeepPrior by adding ResNet layers, data augmentation, and better initial hand localization. [42]

3DCNN is a convolutional network that works with 3D volumetric representations. These 3D volumes are obtained by encoding the depth image using the projective Directional Truncated Signed Distance Function. The network infers the 3D joints directly from these 3D volumes. Their experimental results indicate that CNN benefits from the volumetric representations. [16]

CrossInfoNet decomposes the estimation task into palm pose estimation and finger pose estimation and shares the information between these two subtasks by adopting a two-branch cross-connection structure. They also propose a heat-map guided feature extraction structure to get better feature representation. [11]

2.3 Hand gesture recognition

Gesture recognition is concerned with identifying the hands' postures from images. The applications include human-computer interaction, virtual reality, robotics, and sign language translation.

Gestures can be divided into two categories: static (fine-grained) and dynamic (coarse) gestures. The latter is understood as a series of hand postures through a period. The hand is tracked over this period, and its positions are recorded. The gesture is identified by analyzing these recorded positions. [71] Unlike the dynamic gesture, a static gesture does not consider the movement of the hand but only its shape. This thesis is concerned only with static gestures.

According to Bei Li et al. [30], gesture recognition of static gestures can be divided into three steps: hand segmentation, feature extraction, and gesture classification.

The hand is detected and separated from the complex background during hand segmentation. Previous methods used glove or marker-based methods, as discussed in Section 2.1.3. These methods require additional equipment, which makes them uncomfortable for general usage. Nowadays, computer vision techniques dominate the field of object detection and segmentation. These techniques employ skin color-based or depth-based segmentation or perform background subtraction in the assumption of stationary background.

Feature extraction is the process of extracting key information from data. In the case of gesture recognition, these features include, among others, contour features, palm position, number of fingers, and fingers' lengths. Recently, with the advances in the field of

hand pose estimation, researchers begin to focus on using features extracted from skeletal representations.

Gesture classification methods use the extracted features to recognize gestures. Static gesture classification methods include both generative and discriminative classifiers, such as a support vector machine (SVM), neural network, gaussian mixture model (GMM), and k-nearest neighbor (KNN). Hidden Markov models with dynamic time warping are commonly used for the recognition of dynamic gestures.

The following text shortly presents the feature extraction and gesture classification of two static gesture recognition methods.

The method proposed by S. P. Kasthuri Arachchi et al. [1] trains an SVM model with a combination of 2D and 3D features. The 3D features include finger positions in 3D, the length of fingers from the wrist joint, and fingers' angles. 2D points extracted from the hand's contour and its convex hull together with the ratio of width and height of the bounding box compose 2D features. They achieve relatively low accuracy (93.11 %) in static gesture recognition, considering that they evaluated their system on only six gestures.

Another method from Tong Zhang et al. [69] performs gesture recognition from color images. They train a classifier based on Fuzzy Gaussian Mixture Models (FGMMs), which performs well in rejecting non-gestures with a limited number of training non-gesture samples. They denote the location of k-th joint as $\mathbf{Z}_k = (x_k, y_k)$ in pixel coordinates and the location of the wrist joint as $\mathbf{Z}_1$ for convenience. Then, they subtract the wrist joint from the other joints to ensure that the features are invariant to shifting, transforming the joints to relative positions:

$$\mathbf{Z}_k = \mathbf{Z}_k - \mathbf{Z}_1, k \in \{2, \ldots, 21\}. \tag{2.4}$$

They also ensure invariance to scaling because the distance from the camera to the hand is not fixed. Invariance to scaling in 3D coordinate system with depth images would translate to invariance to different hand sizes. The joint locations are scaled by the maximum norm value to ensure the invariance to scaling:

$$I = \underset{i \in \{2,\ldots,21\}}{\arg\max} \|\mathbf{Z}_i\|_2 \tag{2.5}$$

$$\mathbf{Z}_k = \frac{\mathbf{Z}_k}{\|\mathbf{Z}_I\|_2}, k \in \{2, \ldots, 21\}, \tag{2.6}$$

where $\|\mathbf{Z}_I\|_2$ denotes the Euclidean distance between $\mathbf{Z}_I$ and $\mathbf{Z}_1$. These normalized coordinates are then concatenated into a single array and fed into the FGMM classifier. They achieve an accuracy of 98.06 % on a dataset consisting of seven gesture categories and one non-gesture category.

The idea of relative positions is further developed in Section 5.2.4 where I propose using the relative positions to be transformed into relative distances, which requires computing the relative positions not only between the wrist joint and the other joints but also between all combinations of joints to preserve the relative locations among joints.

Challenges

The following text discusses how gesture classification is affected by scale variation, in-plane and out-of-plane rotation. [60]

Scale variation means that users' hands are of different sizes or that the hands are closer or farther from the camera. The gesture is not dependent on the hand's size. Therefore, scaled versions of the same pose are considered the same gesture.

During in-plane rotation, the hand is rotated in the image plane—neither the pose nor the hand's orientation change during this rotation. Hence, gestures rotated in the image plane are the same gestures, which is an expected behavior from the use-case perspective.

The rotation in world space is called out-of-plane rotation. The pose stays the same, but the hand's orientation changes. Applications, which require a specific orientation, do not consider these gestures as the same.

Chapter 3

Used architectures

The past few decades brought numerous algorithms for solving image processing tasks. Deep neural networks have proved themselves to be especially successful. However, they require large datasets to achieve meaningful results. Nowadays, acquiring these datasets is less problematic, which leads to the constant improvement of neural network architectures.

Deep neural networks were selected for solving hand detection and hand pose estimation. The decision was led by the availability of public datasets and the robustness of deep models.

The following section introduces the YOLOv3 network for depth-based object detection. Then, Section 3.2 describes the JGR-P2O architecture that estimates the pose of the detected hand.

3.1 You Only Look Once

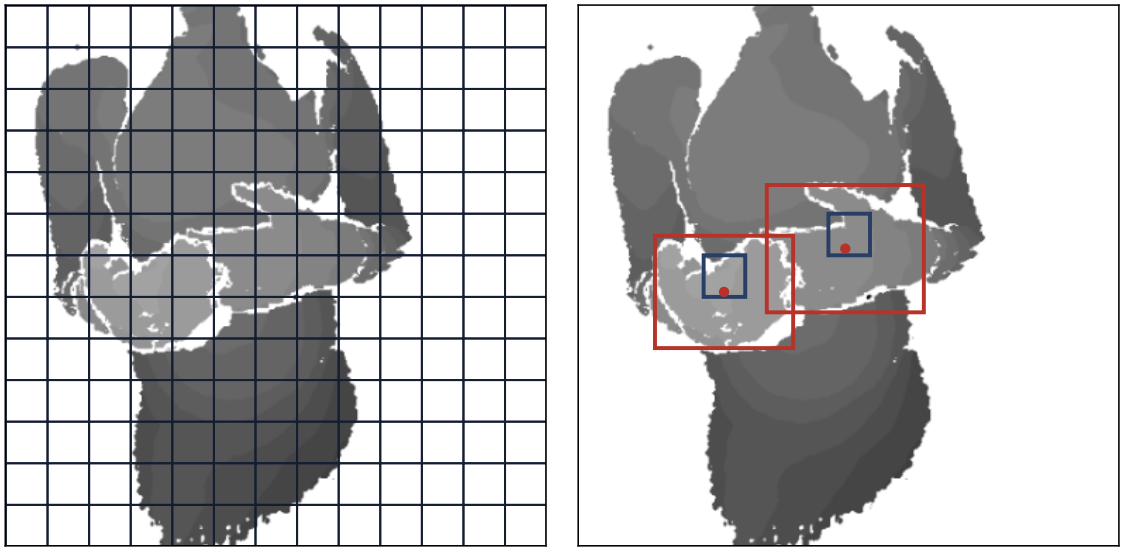
You Only Look Once [46] (YOLO) is a deep neural network for object detection. The first version was introduced in 2015. Since then, the architecture has been modified several times. All versions are generic one-stage object detectors, which were described in section 2.1.2. In 2020, a fourth version was released. Authors applied many techniques used in modern architectures of neural networks, leading to further improvements in both speed and accuracy.

This thesis uses a modified version of the Tiny YOLOv3 [47] to better suit depth images. The YOLOv3 was chosen for its outstanding results in accuracy and speed, as contrasted with two-stage detectors and previous versions of YOLO networks. Both the general and the modified YOLOv3 are described below.

3.1.1 YOLOv3

The following description is based on the original source [47]. The network simultaneously predicts bounding box proposals and probabilities for these boxes. The input image is divided into a grid of cells. The network predicts several bounding boxes for each cell depending on the number of anchors. Figure 3.1 demonstrates the division of the image into the grid of cells.

To better handle the challenge in the object scale mentioned in section 4.1, predictions are made at three different scales. It extracts features similarly to feature pyramid networks [32]. The network first downsamples the image, predicts at the smallest scale, and then upsamples again, combining with features from previous layers, and predicting at larger scales.



(a) (b)

Figure 3.1: **(a)** The YOLO divides the image into a grid of cells, and for each of them, the network makes multiple predictions. **(b)** The figure shows predicted bounding boxes (red) and the responsible cells (blue) along with a red dot representing the center of the bounding box.

The total number of predicted bounding boxes for an image is given by:

$$\sum_{i=0}^{S} C_i^2 \cdot A, \tag{3.1}$$

where S is the number of scales, C_i is the number of cells in a single axis for i-th scale, and A is the number of anchors per cell. The anchors are predefined rough estimates of objects' sizes defined manually in the configuration of the network. In essence, they help the network estimate the size of the detected objects. The YOLO uses three anchors for each scale.

Prediction

Each prediction consists of objectness score, class predictions, and a bounding box proposal.

Objectness score is a probability that a cell contains any object. The probability equals 1 if the anchor of that cell overlaps the object more than any other anchor; in that case, the cell is responsible for that prediction.

Class predictions are probabilities for each object category. The object is classified as the class with the highest probability.

The network predicts four values (t_x, t_y, t_w, t_h) to represent the location of the bounding box. However, they are relative to their respective cells; therefore, they are transformed into global coordinates in a YOLO layer. This process is visualized in Figure 3.2. The sigmoid function σ converts the t_x and t_y coordinates into the range of a single cell. They represent the centroid of the bounding box. Then, the cell offset (c_x, c_y) is added to convert the centroid coordinates into global coordinates. The width t_w and height t_h of the bounding

box is scaled using an anchor box, also known as a bounding box prior (p_w, p_h), which gives the actual size of the bounding box.

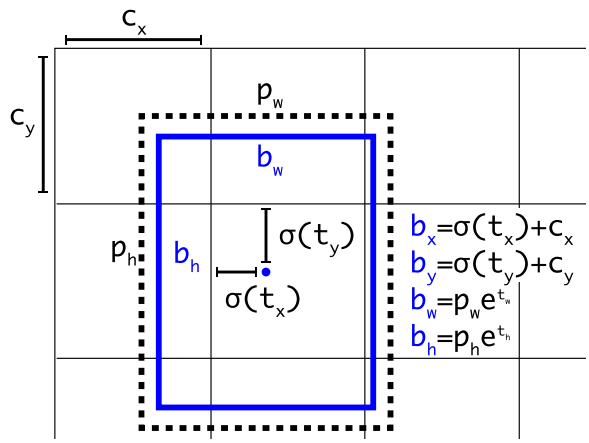


Figure 3.2: The YOLO network predicts four values (t_x, t_y, t_w, t_h) as a representation of the bounding box. The representation is relative to the responsible cell of that prediction. Hence, the YOLO layer transforms the representation into global image representation (b_x, b_y, b_w, b_h) using equations depicted in the figure. It was adopted from the YOLOv3 paper [47].

Most of the predictions are eliminated by non-max suppression [68]. Only bounding box proposals with the highest score (objectness_score · class_probability) remain. Non-max suppression also removes boxes with overlap greater than a certain threshold.

3.1.2 Tiny YOLOv3 for hand detection

Tiny YOLOv3 is simpler, faster, and less accurate than YOLOv3. It predicts in two different scales as opposed to three in YOLOv3. In this work, the network recognizes only a single class—a hand. The general architecture produces the probability of containing an object and its class for each grid box. For a single class, however, the network would produce two probabilities with the same meaning for each grid box. Hence, this specific case is simplified to a single probability per grid box. The architecture of the network is shown in Table 3.1. The number of filters in convolutional layers prior to YOLO layers were decreased to 15 from the previous value of 255 to accept a single class instead of the original 80.

Two loss functions are employed—one for object localization (bounding boxes), and the other one for object classification (objectness score). Intersection over union [51] loss is chosen for bounding boxes and is calculated only for responsible cells. Binary cross entropy is selected to control the objectness score. However, cells that are not responsible for a certain prediction but overlap an object by a certain threshold, which is set to 0.7, are not penalized.

3.2 Joint Graph Reasoning based Pixel-to-Offset Prediction Network

Linpu Fang, Xingyan Liu et al. [13] had thoroughly studied previous works on hand pose estimation from depth images and proposed a new architecture, called Joint Graph Rea-

Layer	Type	Filters	Size / Stride	Input	Output
0	Convolutional	16	$3 \times 3 / 1$	$416 \times 416 \times 1$	$416 \times 416 \times 16$
1	MaxPool	-	$2 \times 2 / 2$	$416 \times 416 \times 16$	$208 \times 208 \times 16$
2	Convolutional	32	$3 \times 3 / 1$	$208 \times 208 \times 16$	$208 \times 208 \times 32$
3	MaxPool	-	$2 \times 2 / 2$	$208 \times 208 \times 32$	$104 \times 104 \times 32$
4	Convolutional	64	$3 \times 3 / 1$	$104 \times 104 \times 32$	$104 \times 104 \times 64$
5	MaxPool	-	$2 \times 2 / 2$	$104 \times 104 \times 64$	$52 \times 52 \times 64$
6	Convolutional	128	$3 \times 3 / 1$	$52 \times 52 \times 64$	$52 \times 52 \times 128$
7	MaxPool	-	$2 \times 2 / 2$	$52 \times 52 \times 128$	$26 \times 26 \times 128$
8	Convolutional	256	$3 \times 3 / 1$	$26 \times 26 \times 128$	$26 \times 26 \times 256$
9	MaxPool	-	$2 \times 2 / 2$	$26 \times 26 \times 256$	$13 \times 13 \times 256$
10	Convolutional	512	$3 \times 3 / 1$	$13 \times 13 \times 256$	$13 \times 13 \times 512$
11	Convolutional	1024	$3 \times 3 / 1$	$13 \times 13 \times 512$	$13 \times 13 \times 1024$
12	Convolutional	256	$1 \times 1 / 1$	$13 \times 13 \times 1024$	$13 \times 13 \times 256$
13	Convolutional	512	$3 \times 3 / 1$	$13 \times 13 \times 256$	$13 \times 13 \times 512$
14	Convolutional	15	$1 \times 1 / 1$	$13 \times 13 \times 512$	$13 \times 13 \times 15$
15	**YOLO**	-	-	$13 \times 13 \times 15$	$13 \times 13 \times 3 \times 5$
16	Route 13	-	-	-	$13 \times 13 \times 512$
17	Convolutional	128	$1 \times 1 / 1$	$13 \times 13 \times 512$	$13 \times 13 \times 128$
18	Upsample	-	$2 \times 2 / 1$	$13 \times 13 \times 128$	$26 \times 26 \times 128$
19	Route 18, 8	-	-	-	$26 \times 26 \times 384$
20	Convolutional	256	$3 \times 3 / 1$	$26 \times 26 \times 384$	$26 \times 26 \times 256$
21	Convolutional	15	$1 \times 1 / 1$	$26 \times 26 \times 256$	$26 \times 26 \times 15$
22	**YOLO**	-	-	$26 \times 26 \times 15$	$26 \times 26 \times 3 \times 5$

Table 3.1: The Tiny YOLOv3 predicts at two scales as indicated by the two YOLO layers. Features from the image are extracted using a series of convolutional and max-pooling layers. The first YOLO layer predicts at 13×13 scale, which is followed by image upsampling, more convolutional layers, and by the second YOLO layer at 26×26 scale. The route layer brings previous layers together with a concatenation of features of those layers.

soning based Pixel-to-Offset Prediction Network (JGR-P2O), that surpasses all others in terms of accuracy. This section is based on the JGR-P2O paper [13].

The JGR-P2O works as a dense estimator of the hand's skeleton. It takes a depth image as the input and outputs the joints' positions in the image plane (UV coordinates) and depth space (Z coordinate). Its architecture is shown in Figure 3.3. The JGR-P2O operates on images with a resolution of 96×96 pixels. The JGR-P2O architecture starts with a 7×7 convolutional layer with stride 2, followed by a residual module and max-pooling layer to decrease the resolution from 96 to 24. Then, another two residual modules follow, preceding an hourglass convolutional network [41], which is a high-efficient network used as the backbone to extract intermediate local feature representation. Then, the Joint Graph Reasoning module models the joints' connections of the hand's skeleton, creating an enhanced intermediate representation. Next, the pixel-to-offset prediction module estimates the offsets from pixels to joints for each coordinate. The final joints' locations are retrieved by averaging the offsets.

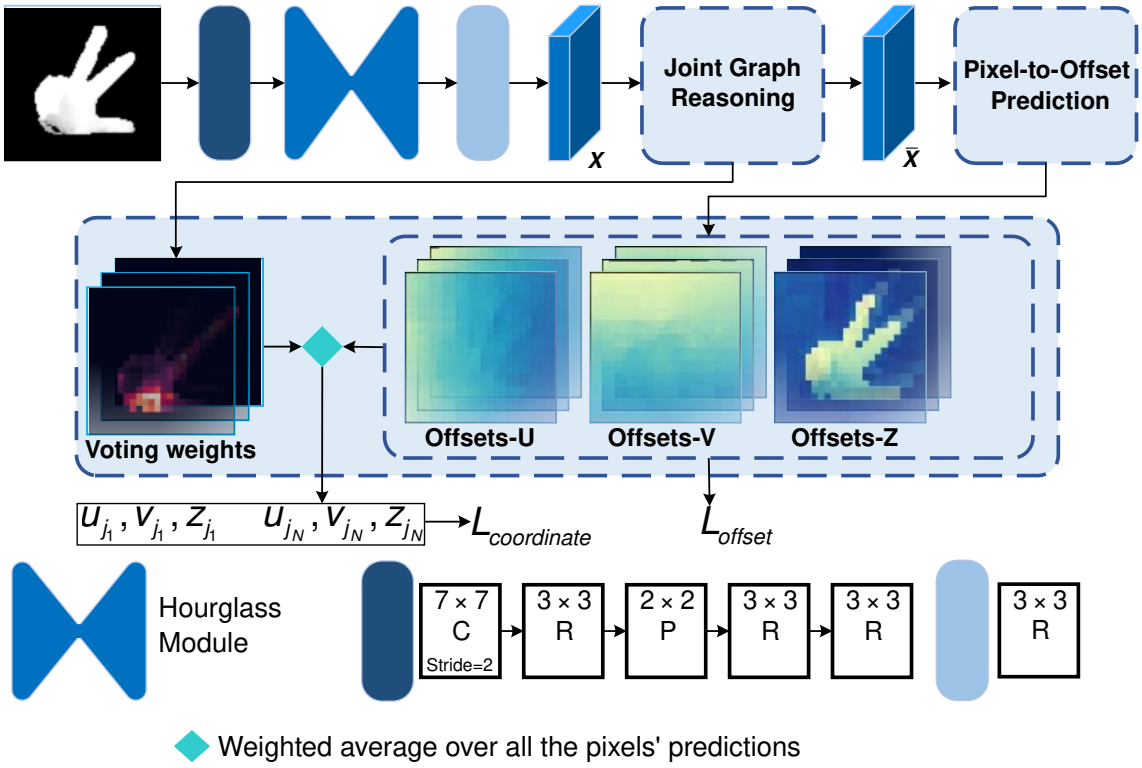


Figure 3.3: The JGR-P2O architecture consists of three stages. First, feature representation $\mathbf{X}$ is extracted from the input depth image in the Hourglass module. Then, the Joint Graph Reasoning module enhances the representation using information about adjacent joints, transforming it into enhanced feature representation $\widetilde{\mathbf{X}}$. The last module, Pixel-to-Offset, produces joint offsets in each pixel. The authors stack two layers of these modules to slightly improve performance. In the figure, C, R, and P denote a convolutional, a residual, and a pooling layer. The figure was adopted from the JGR-P2O paper [13].

Residual Block

Kaiming He et al. [26] introduced a residual block to ease the training of substantially deep networks. They formally define the residual block as:

$$\mathbf{y} = F(\mathbf{x}, \{\mathbf{W}_i\}) + \mathbf{x}, \tag{3.2}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the considered layers. The function $F(\mathbf{x}, \{\mathbf{W}_i\})$ is a residual mapping to be learned. The operation $F + \mathbf{x}$ is a shortcut connection with an element-wise addition. The weight layers $\mathbf{W}_i$ are usually convolutional. Figure 3.4 illustrates the residual block in the JGR-P2O network.

3.2.1 Hourglass Network

The hourglass network [41] is a convolutional neural network for the task of pose estimation. Features are computed at different scales, motivated by the need to identify features from tiny details such as fingertips to the overall pose standing as a cohesive unit made of smaller parts. The hand's orientation, the fingers' arrangement, and the relationship between adjacent joints are better recognized at different scales.

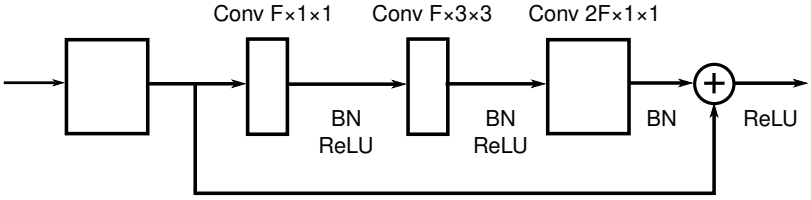


Figure 3.4: A schema of the residual block used in JGR-P2O. Conv, BN, ReLU stands for convolutional layer, batch normalization layer, and rectified linear unit as an activation function. The F detones the number of features used.

The hourglass computes features at multiple scales and combines them back together to produce an enhanced pixel-wise representation. The architecture is created by combining top-down and bottom-up processing. The top-down sequence is composed of residual blocks and max-pooling layers until the feature representation reaches its smallest resolution. Then, the bottom-up processing uses residual blocks followed by upsampling layers and combinations of features across scales until the image representation reaches the hourglass network's input resolution.

3.2.2 Joint Graph Reasoning Module

The joint graph reasoning module (JGR), depicted in Figure 3.5, augments the intermediate local feature representation for each pixel. It uses the extracted features from the backbone network to generate joints' features using a pixel-to-joint voting mechanism. Next, the graph reasoning mechanism models the joints' dependencies using a joint-to-joint undirected graph specifying adjacent joints. The mechanism enhances the joints' features. These enhanced features are mapped back to the pixels' representation to obtain pixel-wise joint context representation. Finally, the original features from the backbone are combined with the joint context representation.

Pixel-To-Joint Voting

The idea is that each joint has its informative pixels. This idea is further developed into a voting mechanism for retrieving joints' features by aggregating pixel values using trainable voting weights. Given features from the backbone $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$, the voting weights $\mathbf{W} \in \mathbb{R}^{N \times H \times W}$ (H, W, C, N denote height, width, number of channels, and number of joints) are computed as:

$$\mathbf{W} = \Phi(\phi(\mathbf{X})), \tag{3.3}$$

where ϕ is a 1×1 convolution, Φ is spatial softmax normalization. Then features $\mathbf{F} \in \mathbb{R}^{N \times C}$ of all joints are computed as a matrix multiplication representing a weighted average over the transformed pixel-wise representation:

$$\mathbf{F} = \mathbf{W} \cdot \varphi(\mathbf{X}), \tag{3.4}$$

where φ is a 1×1 convolution.

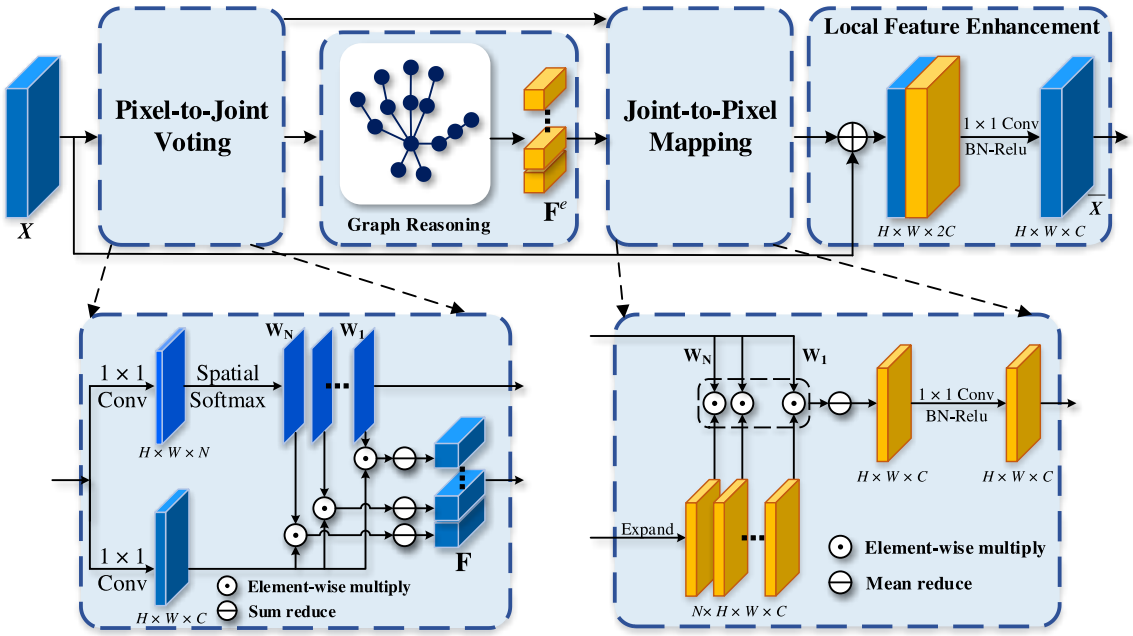


Figure 3.5: Joint Graph Reasoning module enhances feature representation $\mathbf{X}$ by explicitly modeling the dependencies among joints. First, the Pixel-to-Joint Voting mechanism converts the input feature representation into joints' feature representation $\mathbf{F}$. This representation is altered into evolved features $\mathbf{F}^e$ by Graph Reasoning mechanism by modeling the dependencies among joints. The Joint-to-Pixel Mapping transforms the joints' features back into pixel-wise representation, which are combined with original feature representation $\mathbf{X}$ in Local Feature Enhancement mechanism, creating enhanced feature representation $\widetilde{\mathbf{X}}$. The figure was adopted from the JGR-P2O paper [13].

Graph Reasoning

Given joints' features $\mathbf{F}$ from the pixel-to-joint voting mechanism are augmented by modeling the dependencies among joints creating evolved joints' features $\mathbf{F}^e$:

$$\mathbf{F}^e = \sigma(\mathbf{A}^e \mathbf{F} \mathbf{W}^e), \tag{3.5}$$

where σ is a ReLU function, $\mathbf{A}^e \in \mathbb{R}^{N \times N}$ is a matrix modeling dependencies among joints, and $\mathbf{W}^e \in \mathbb{R}^{C \times C}$ is a trainable transformation matrix, which can be implemented as a dense layer. The authors compared three different methods to model dependencies among joints $\mathbf{A}^e$. They received the best results with a skeleton graph method:

$$\mathbf{A}^e = \widetilde{\mathbf{D}}^{-\frac{1}{2}} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{3.6}$$

where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is an adjacency matrix $\mathbf{A}$ of the hand skeleton with self connections represented as an identity matrix $\mathbf{I}_N$. Adjacent and non-adjacent joints in the adjacency matrix are indicated by ones and zeros respectively. $\widetilde{\mathbf{D}}$ is a degree matrix defined as follows: $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$. In graph theory, a degree matrix is a diagonal matrix containing information about the number of edges attached to each vertex.

Joint-To-Pixel Mapping

The evolved joints' features $\mathbf{F}^e$ are transformed back to pixel-wise joint context representation by doing a reverse operation of pixel-to-joint voting. First, the context representation of joint k is computed as: $\mathbf{c}_{ik} = w_{ik}\mathbf{f}_k^e$, where w_{ik} is the mapping weight from joint k to pixel p_i from Equation (3.4), and $\mathbf{f}_k^e$ is the evolved feature of joint k. Then, the final pixel-wise joint context representation for pixel p_i is the mean over the context representation of all joints:

$$\mathbf{c}_i = \rho(\frac{1}{N}\sum_k \mathbf{c}_{ik}), \tag{3.7}$$

where ρ is a 1×1 convolution with BN and ReLU. All joint context representations $\mathbf{c}_i$ constitute $\mathbf{C} \in \mathbb{R}^{H \times W \times C}$.

Local Feature Enhancement

Finally, local feature enhancement concatenates the pixel-wise joint context representations $\mathbf{C}$ from the graph reasoning module with the features $\mathbf{X}$ from the backbone:

$$\widetilde{\mathbf{X}} = \tau([\mathbf{C}, \mathbf{X}]), \tag{3.8}$$

where τ is a 1×1 convolution with BN and ReLU serving as a transformation function for fusing these representations, resulting in a combined feature map $\widetilde{\mathbf{X}}$ that is used next in the pixel-to-offset prediction module.

3.2.3 Pixel-To-Offset Prediction Module

The depth image consists of image plane coordinates (UV coordinates) with depth information (Z coordinate). Instead of predicting these coordinates directly, the pixel-to-offset prediction module produces an offset map containing offsets from pixels to each joint coordinate. Specifically, this module is implemented by a 1×1 convolution layer taking the combined feature map $\widetilde{\mathbf{X}}$ as input and outputs $3N$ offset maps—one offset map for each coordinate of each joint.

Then, the UVZ coordinates $(u_{j_k}, v_{j_k}, z_{j_k})$ are retrieved as a weighted average over all offsets in the offset map:

$$\begin{aligned} u_{j_k} &= \sum_i w_{ki}(u_{p_i} + \Delta u_{ki}) \\ v_{j_k} &= \sum_i w_{ki}(v_{p_i} + \Delta v_{ki}) \\ z_{j_k} &= \sum_i w_{ki}(z_{p_i} + \Delta z_{ki}) \end{aligned}, \tag{3.9}$$

where $(u_{p_i}, v_{p_i}, z_{p_i})$ denote UVZ coordinates of pixel p_i, and $(\Delta u_{ki}, \Delta v_{ki}, \Delta z_{ki})$ predicted offsets from pixel p_i to joint k. Voting weight w_{ki} from the pixel-to-joint voting mechanism 3.2.2 signifies the importance of pixel p_i in predicting the location of joint k.

3.2.4 Loss functions

Two loss functions are used during training. The primary one is a coordinate-wise regression loss $L_{coordinate}$, supervising the difference between the predicted coordinates and the ground

truth coordinates. The secondary loss function L_{offset} is a pixel-wise offset regression loss, supervising the offsets from which the coordinates are computed.

The coordinates are computed by Equation (3.9). The $L_{coordinate}$ function is defined as the Huber loss [27] between the predicted and the ground truth coordinates:

$$L_{coordinate} = \sum_{k} \sum_{c} L_{\delta}(c_{j_k} - c^*_{j_k}), \tag{3.10}$$

where c_{j_k} is one of the predicted coordinates of joint k, and $c^*_{j_k}$ is the corresponding ground-truth coordinate. The authors chose the Huber loss L_{δ}, as it is less sensitive to outliers than the mean squared error loss.

Furthermore, the L_{offset} loss supervises the generation of offsets, which slightly improves the model's final performance. Similarly to the primary loss, it is computed as the Huber loss between the predicted and the ground truth offsets:

$$L_{offset} = \sum_{k} \sum_{i} \sum_{c} L_{\delta}(\Delta c_{ki} - \Delta c^*_{ki}), \tag{3.11}$$

where c_{ki} is the predicted offset value from pixel p_i to joint k for one of the coordinates, and c^*_{ki} is the corresponding ground truth offset value.

Chapter 4

Datasets

The datasets are the most crucial aspect of hand pose estimation and related tasks. It is vital to choose a dataset that suits the target use case the most. The first section describes the many challenges of the real environment. These are followed by sections providing an overview of existing datasets. Some of them are used further in this thesis and will be described in greater detail. Finally, the last section provides details on a custom dataset created for evaluation of the proposed system.

4.1 Challenges in hand position determination

Certain applications present different challenges. The ability of the model to overcome these challenges is primarily dependent on the dataset. Generally, the more challenges the dataset addresses, the more robust solution is reached. Barsoum [3] presented several of these challenges that may need to be addressed by algorithms and datasets in hand detection and pose estimation tasks. The following text shortly describes them.

Hand shape

Every person has a different shape of hands. Although the shape does not play a significant role in hand position determination as other challenges discussed in this section, it could still affect the outcome. Choosing datasets with multiple subjects may help overcome this challenge.

Self-occlusion

Self-occlusion occurs when a hand blocks the view on some of its parts. Depending on its pose, several fingers or the palm can be occluded. The datasets should include hands captured with a large variation of articulations and viewpoints.

The number of visible joints in the datasets of the HANDS17 challenge is depicted in Figure 4.1. They also showed that the models achieve lower performance with higher occlusion, making object interaction even more difficult.

Object grabbing

Object grabbing is one of the most challenging problems in hand pose estimation. Because hands can hold many different objects, the datasets need to be extensive. This problem is not considered because of the target use case of this work.

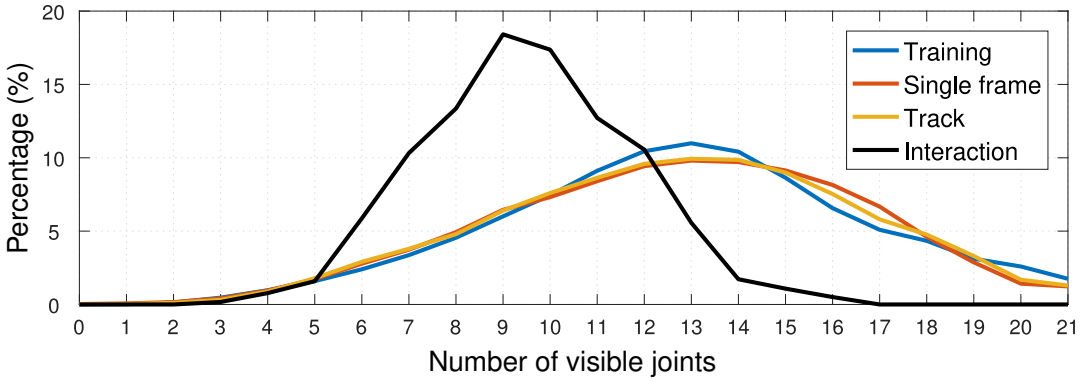


Figure 4.1: The figure was adopted from the HANDS17 challenge [65], depicting the number of visible joints in the datasets they used for training and testing. The testing dataset is divided into single frame estimation, object tracking, and object interaction, which is particularly difficult due to high occlusion.

Multiple hands

There will be situations when more than a single hand is present in the image. The use case may necessitate a proper selection of algorithms and datasets for both object detection and hand pose estimation depending on its requirements.

Scale

Hands in the image can be closer or farther and have different proportions to other parts of the image. This problem necessitates a careful selection of algorithms and datasets.

Gloves

The model can perform poorly if the user wears gloves. However, this work's target use case does not allow wearing gloves, so it is not further considered.

4.2 Hand segmentation datasets

This section covers datasets of depth images, which can be used for hand segmentation. Moreover, bounding boxes can be easily produced for training hand detection models because these datasets are pixel-wise labeled.

Table 4.1 provides an overview of four datasets. The HandSeg dataset is described in more detail in the following section.

Dataset	Annotations	Frames	Subjects	Sensor	Resolution
Freiburg [73]	synthetic	43,986	20	Unreal Engine	320 × 320
HandNet [59]	heuristic	212,928	10	RealSense SR300	320 × 240
NYU [56]	automatic	6,736	2	Kinect v1	640 × 480
HandSeg [39]	automatic	158,315	10	RealSense SR300	640 × 480

Table 4.1: Overview of hand segmentation datasets adopted from the HandSeg paper [39].

4.2.1 HandSeg

The HandSeg dataset consists of 158,315 depth images of size 640x480. The images were captured with a RealSense SR300 camera and automatically annotated with labels as single-channel masks where a nonzero pixel indicates a hand. Most images contain both hands, which makes the dataset suitable for training models to distinguish between them. In this work, the dataset is used for its size suitable for training deep models. The dataset contains three female and seven male subjects, and all images were captured in the same scene. [39] Figure 4.2 shows sample images and their respective annotations.

In most cases, the images have no background except for a human body, resulting from the camera's technology for capturing images. This camera's operating range is 0.3–2 meters, and, as a result, no farther objects are visible. The lack of background in the training dataset might cause the final model to struggle in scenes with a rich background comprising small objects. If such behavior is undesirable in the target use case, training on an additional dataset with a rich background will help the model to overcome this challenge.

Although this dataset was annotated for hand segmentation, the hand detector in this work requires bounding boxes for input. Hence, from the mask annotations were generated new annotations in the form of bounding boxes.

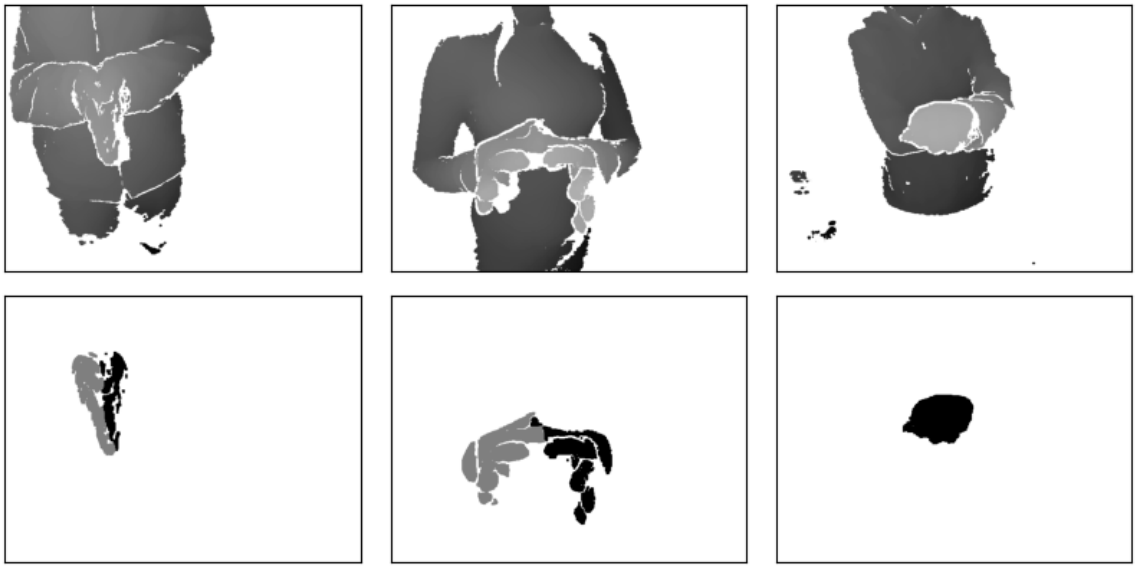


Figure 4.2: The HandSeg dataset contains 158,315 depth images with annotations in the form of a single channel mask. The first row contains depth images, and the second one their respective masks. The annotations differentiate between the left hand (black) and the right hand (grey).

4.3 Hand pose estimation datasets

Suspancic et al. [28] demonstrate that the realism and diversity of a dataset are crucial to the performance of a hand pose estimation method and that the dataset is as crucial as the choice of the model. That is the reason so many datasets were created in the past decade.

Table 4.2 was adapted from a GitHub repository from xinghaochen [61] that provides a complete list of hand pose estimation resources. Most of these datasets include skele-

tal annotations—usually 21 joints for each hand. However, the Hands in Action and the SynHandEgo datasets use mesh annotations instead.

Hand pose estimation poses many challenges, as described in Section 4.1, including object grabbing. The two datasets—FHAD and Hands in Action—consist of images with hands holding some objects. Object grabbing was tackled in competitions HANDS17 [65] and HANDS19 [2].

All datasets from Table 4.2 were captured using either Time of Flight (ToF) depth cameras (e.g., Intel RealSense SR300, Microsoft Kinect, Creative Interactive Gesture) or structured light cameras (e.g., ASUS Xtion Pro). The ToF cameras measure the round-trip time it takes for the emitted photons to return to the sensor. A structured light camera, on the other hand, projects a specific pattern onto the scene. The way the pattern is deformed is used to calculate the depth. The most important difference in depth cameras is their operating range. The SR300 operates at 0.2–1.5 m, which makes them suitable for applications on short distances. On the other hand, a D415 camera's maximum range is 10 m. Another crucial property is resolution, which is usually 640×480 or 320×240 pixels.

Dataset	Year	Type	Joints	View	Subjects	Frames
ASTAR [63]	2013	Real	20	3rd	30	870
NYU [55]	2014	Real	36	3rd	2	72k/8k
ICVL [54]	2014	Real	16	3rd	10	331k/1.5k
MSRA14 [45]	2014	Real	21	3rd	6	2,400
Hands in Action [57]	2014	Real	-	3rd	-	-
MSRA15 [52]	2015	Real	21	3rd	9	76,375
MSRC (FingerPaint) [49]	2015	Synthetic	21	both	1	100k
HandNet [59]	2015	Real	6	3rd	10	202k/10k
BigHand2.2M [66]	2017	Real	21	3rd	10	2.2M
FHAD [14]	2018	Real	21	ego	6	100k
SynHand5M [38]	2018	Synthetic	-	3rd	-	5M
SynHandEgo [37]	2019	Real	-	ego	-	-

Table 4.2: Overview of depth-based hand pose estimation datasets adopted from the xinghaochen's GitHub repository [61].

The ICVL, MSRA, and NYU datasets are traditional datasets used for comparison of hand pose estimation models. The authors of the BigHand dataset showed that these datasets have low variation in viewpoint and hand articulation, as shown in Figure 4.3. This low variation results in an overall worse performance of the trained model, as clearly indicated in Table 4.3, where the BigHand dataset's authors made a cross-benchmark comparison of four datasets (BigHand, ICVL, NYU, MSRC).

4.3.1 MSRA15

The MSRA15 dataset is a collection of 76,500 depth images captured from nine subjects. They used Intel's Creative Interactive Gesture Camera, which uses the Time-of-Flight technology for capturing images. The annotations were made semi-automatically using an optimization method that was run iteratively with manual correction after each iteration. The dataset consists of 17 hand gestures. Each subject was asked to move their hand rapidly before the camera; 500 images were recorded each time. The authors chose gestures mainly

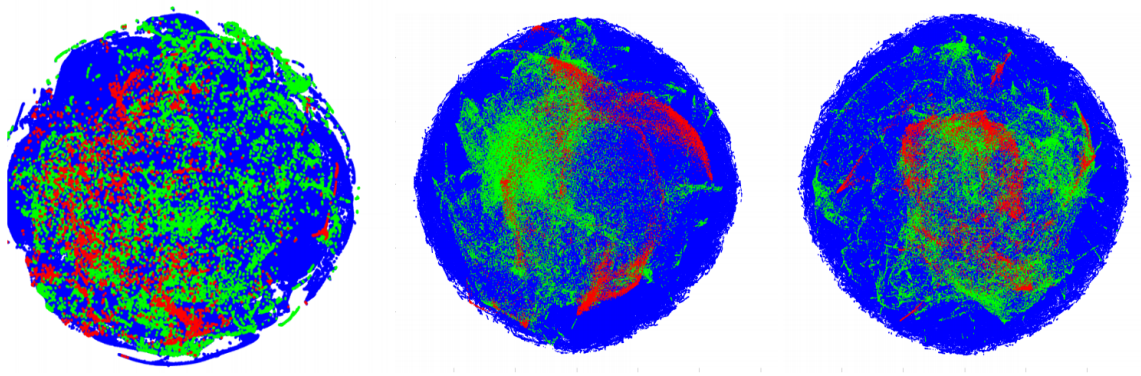


Figure 4.3: The BigHand paper [66] provides a comparison of articulation and viewpoint spaces of BigHand, NYU, ICVL datasets using an unsupervised dimensionality reduction technique called t-SNE embedding [35]. BigHand is represented by the blue dots, NYU by green dots, and ICVL by red dots. The figures show coverage comparison of global viewpoint space (left), articulation angle space (middle), and hand angle space (right).

Train \ Test	ICVL	NYU	MSRC	Bighand
ICVL	**12.3**	35.1	65.8	46.3
NYU	20.1	**21.4**	64.1	49.6
MSRC	25.3	30.8	**21.3**	49.7
BigHand	**14.9**	**20.6**	**43.7**	**17.1**

Table 4.3: The table, adopted from the BigHand paper [66], depicts a cross-benchmark comparison of four datasets using Holi CNN model [64]. Each cell contains an average error in millimeters. The model achieves the best performance when trained and tested on the same dataset or trained on the BigHand dataset.

from the American Sign Language to span the finger articulation space as much as possible. They also show that their dataset has more considerable viewpoint variations than previous datasets—yaw covers almost the whole [-90, 90] range and pitch [-10, 90] degrees. [52] Figure 4.4 lists a few images from this dataset along with their annotations.

The images are highly preprocessed. They contain no background and have arms removed, making them too distinct from real images. The images have to be preprocessed to be similar to those from the MSRA, or another dataset has to be used to train a hand pose estimation model.

This dataset was used to demonstrate the feasibility of using 3D hand pose information to perform gesture recognition from Section 5.2.4 solely by comparing the joints' distances.

4.3.2 BigHand

The BigHand dataset consists of 2.2 million images with significant variation in hand articulation and viewpoint captured on 12 subjects. A few images, along with their annotations, are depicted in Figure 4.5. As contrasted with the MSRA15 dataset from the previous section, the BigHand contains images similar to real images with no preprocessing performed

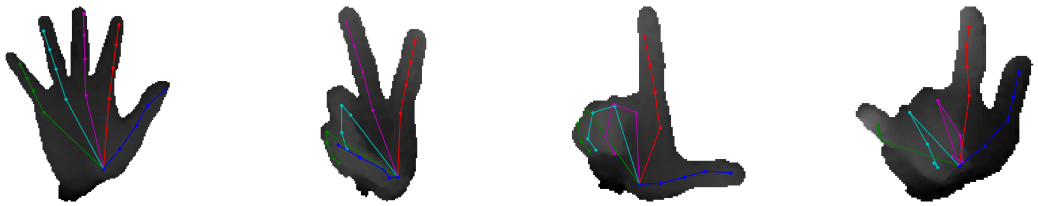


Figure 4.4: Sample depth images together with ground truth annotations showing four different gestures from the MSRA15 dataset for hand pose estimation.

on them. The dataset was captured with an Intel RealSense SR300 camera. It consists of three image types: schemed, random, and egocentric poses.

The schemed poses are aimed to fully cover the articulation space. The authors chose 32 extremal poses such that each finger is fully extended or bent. They captured the transition between each pair of these extremal poses for maximum coverage of the articulation space. These poses make up the majority of the dataset—a total of 1.534 million frames.

The subjects were asked to perform arbitrary poses to cover the pose space. These random poses make up 375 thousand frames.

A total of 290 thousand frames were captured in an egocentric view performing 32 extremal poses with random movements.

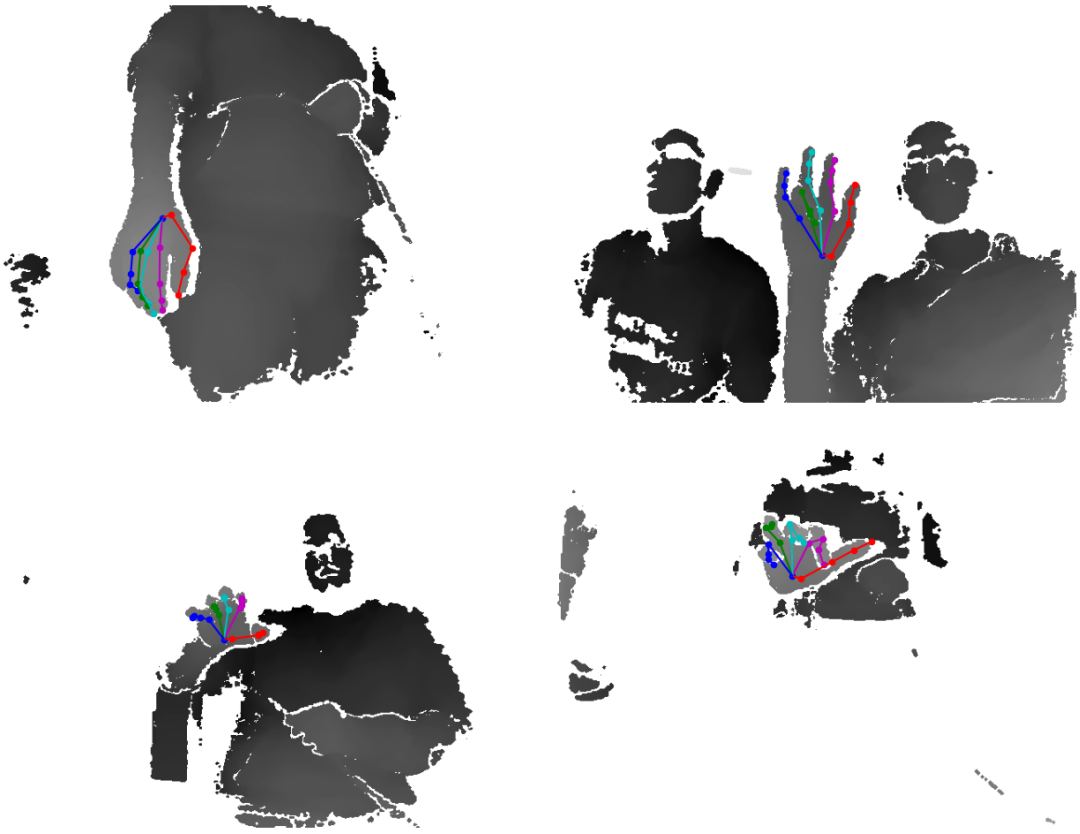


Figure 4.5: Images in the BigHand are not preprocessed, and consequently resemble images from a real environment containing background. The figure depicts four images together with their corresponding skeletal ground truth annotations.

4.4 A custom dataset

The final system has to be verified in a real environment. The dataset should help confirm that all stages—hand detection, hand pose estimation, and gesture recognition—work together as expected, especially for the target use case. The dataset includes variations in hand poses and hand size, varying distance from the camera, and rich background.

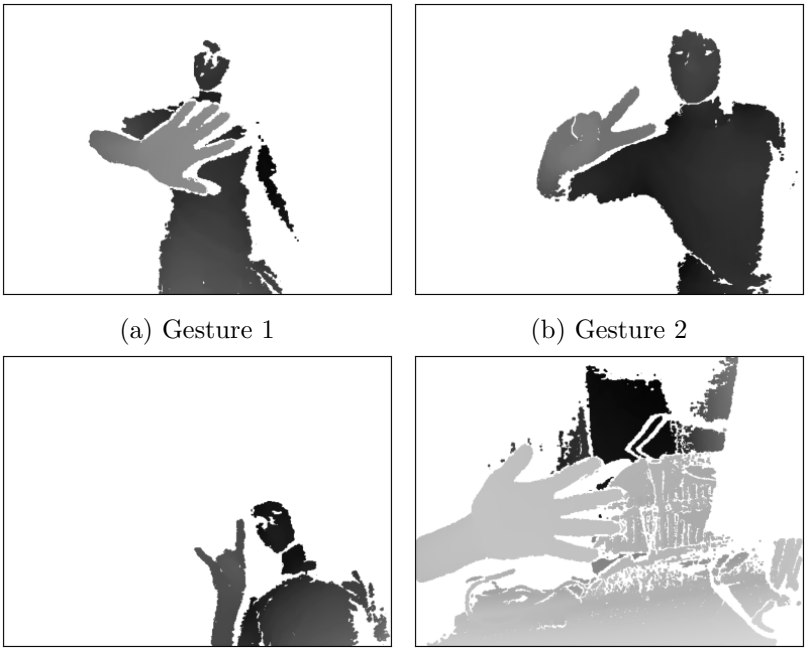
The dataset was captured using an SR305 camera[1] with a maximum capturing distance of 1.5 meters. The resolution of images is 640×480 pixels, which is similar to most datasets introduced in previous sections. The dataset consists of 35 sequences, each targeting a specific scenario. The sequences are tens to hundreds of frames long. Most of them were captured at 10 or 20 FPS. A total of 4307 frames were captured from four subjects. Invalid frames were discarded. Each sequence was manually annotated with a gesture label, flags indicating the presence of the left or right hand, and a subject's number. Three labels are used: *Gesture 1* (opened palm with fingers outstretched and apart), *Gesture 2* (number two), and *Non-gesture*. Table 4.4 summarizes the number of sequences and frames. Sample images are displayed in Figure 4.6.

Label	Sequences	Frames
Gesture 1	12	1116
Gesture 2	8	518
Non-gesture	14	2673

Table 4.4: The custom dataset was created to evaluate the final gesture recognition system. The dataset contains 4307 frames divided into three classes: *Gesture 1*, *Gesture 2*, and *Non-gesture*.

For completeness, the principal point and the focal length are $(313.683, 242.755)$ and 476.007 respectively. The depth unit of the depth sensor is 0.125 mm.

[1] https://www.intelrealsense.com/depth-camera-sr305/



(a) Gesture 1 (b) Gesture 2

(c) Non-gesture (d) Gesture 1 with background

Figure 4.6: The custom dataset was created specifically to evaluate the target use-case of this work. It contains two gestures in the various background and with a distance ranging from 0.2 to 1.5 meters. Many *non-gesture* poses were included to verify that invalid gestures are not accepted.

Chapter 5

Design of gesture acceptance system

This chapter proposes a solution for gesture recognition from skeletal information of the hand and incorporates it in a system for accepting gestures. Given an image to the system, it has to decide whether the hand makes a user-specified gesture. As further discussed, the system is composed of three main stages. First, the hand is detected in the given depth image, producing an image subregion. This subregion is necessary for the second stage, which is hand pose estimation. This stage produces a set of 3D coordinates called keypoints representing the hand's skeleton. Then, the gesture acceptance module uses this information to recognize gestures. The system's requirements are discussed in Section 5.1, which is followed by Section 5.2 proposing the gesture recognition system along with its limitations.

5.1 System requirements

This section presents the system's requirements, which considerably affect its design. These requirements include a generic solution, a real-time application, and feedback for the user.

Generic solution

Many gesture recognition approaches train a parametric model to classify a specific set of gestures. These solutions require large enough datasets for their training, which makes them less portable to other applications. This work, however, aims for a flexible solution capable of being adopted in many applications. Each of them may require a different gesture. Therefore, the target gesture is not built into the system, but the user specifies the gesture instead.

Real-time application

Many possible applications require that the system is running in real-time. Fortunately, researchers in object detection and pose estimation made significant progress to make efficient and accurate systems. Nonetheless, we still have to make conscious decisions to choose suitable models for these tasks.

Feedback on the gesture

The user should receive feedback for each of their fingers. If the pose is not correct, the system should display which fingers are improperly placed.

5.2 Proposed design

Section 2.3 presented that static gesture recognition systems are composed of hand segmentation, feature extraction, and gesture classification. This work utilizes a generic object detector performing hand detection as the first step, as will be further discussed in Section 5.2.1. The step is finished with a preprocessing proposed in Section 5.2.3, which removes everything in the image but the hand, producing a result equivalent to image segmentation. Section 5.2.2 presents the feature extraction, which is accomplished by estimating the hand's pose represented by the joints' positions of the hand's skeleton. The skeletal-based approach was chosen mainly to satisfy the generic usage requirement described in Section 5.1. The computation of the actual features from the hand's skeleton is presented in Section 5.2.4 along with the description of the gesture classification method.

Because the user selects the target gesture, the system's usage consists of two steps. The first step can be understood as a setup stage. During this step, the user captures depth images representing the target gesture. Hand poses are extracted from these images using hand detection and hand pose estimation stages. They are stored in a database of registered poses representing the target gestures. The second step is the actual usage of the system. The features are extracted from the input depth image and are compared to registered poses in the gesture acceptance stage. The process of the system's usage is depicted in Figure 5.1.

5.2.1 Hand detection stage

The process starts with a camera capturing depth images. These are passed further to the hand detection stage, responsible for detecting a hand in the given image. These images are usually not in the form suitable for the model of choice. Thus, they are preprocessed. Because the detector expects square images as the input, the images have to be padded or cropped and then resized to the expected size. After that, normalization of pixel values to the [0, 1] range is performed to boost the model's performance.

In object detection, the efficiency of one-stage detectors is great enough to satisfy the requirement of running real-time. The family of YOLO detectors can detect objects at 40–160 FPS, which is mainly dependent on the backbone architecture. I chose the Tiny YOLOv3 since its backbone is tiny, and its performance is still satisfactory. The architecture has about eight million trainable parameters. This detector was described in Section 3.1.

It outputs bounding boxes and scores for every grid cell. Non-max suppression selects only a few of these predictions that are above a certain overlap and score threshold. [68] There may be more than one prediction, but only a single hand can be passed to the next stage because the final system is expected to return a single answer. Hence, only the hand with the highest probability is selected and passed to the next stage.

5.2.2 Hand pose estimation stage

The hand pose estimation stage's goal is to determine the hand's skeleton by precisely estimating the joints' locations in 3D space. It receives the original depth image with

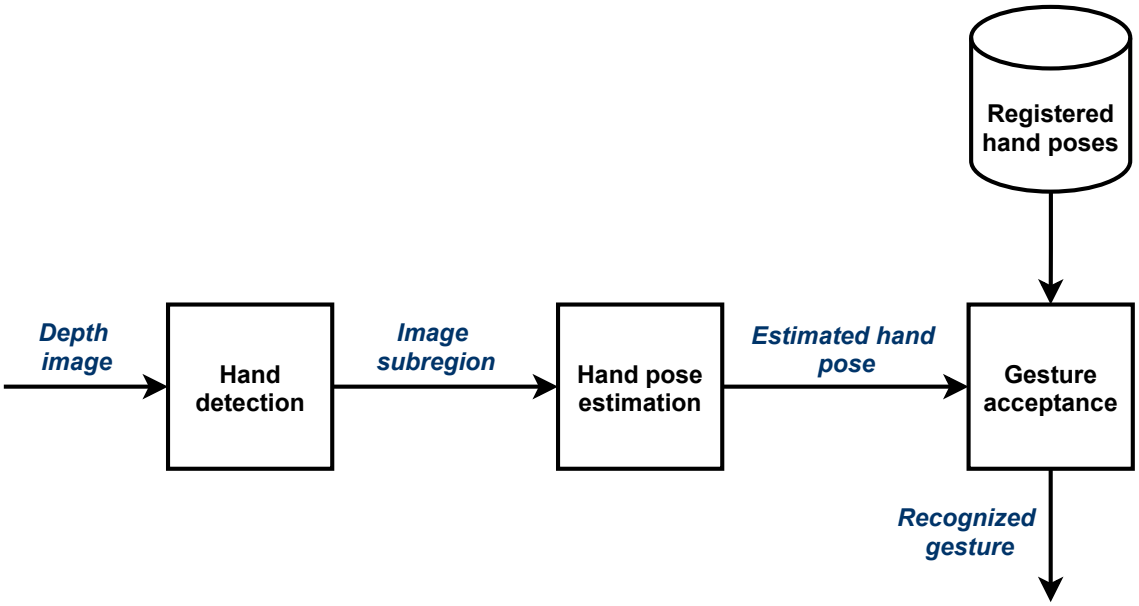


Figure 5.1: The flowchart depicts the main components of the proposed system. The hand detection stage takes a depth image as input and produces an image subregion containing the detected hand. Then, the hand pose estimation infers a pose from this image and passes it on to the gesture acceptance stage, which compares the input pose with a local database of registered poses, resulting in a decision of whether the input pose agrees with some of the registered ones.

a bounding box defining the area containing the hand. The image goes through a series of preprocessing steps before it is passed to a hand pose estimator, which will be described in Section 5.2.3. The image is cropped to the bounding box, from which the background is removed, and then it is resized to the expected input size of the estimator with the depth normalized to the [0, 1] range.

I chose the state-of-the-art JGR-P2O network for this stage for both its accuracy and speed. The authors claim that it runs at 110 fps on a single NVIDIA 1080Ti GPU. Moreover, its architecture is relatively small in terms of trainable parameters, of which there are only 1.4 million. [13] The JGR-P2O does not predict the coordinates in 3D space but rather estimates the pixel coordinates along with a Z coordinate of the world coordinates. These UVZ coordinates are then converted to XYZ world coordinates. This process was described in Section 2.2.2

Experimentation with the trained model showed that its performance is low if the images contain noise or if the depth is not normalized. Hence, a pipeline consisting of several preprocessing steps was developed. The following section describes the pipeline in more detail.

5.2.3 Preprocessing for estimation stage

Experimentation with the implemented JGR-P2O model showed that complex preprocessing is essential for determining a precise pose. Real images contain background and various noise that have to be dealt with. In addition, as shown further in this section, better performance is achieved using a 3D crop as opposed to the usual 2D crop.

To prepare the depth image for hand pose estimation, it must go through the following preprocessing steps:

1. cropping to a bounding box
2. performing Otsu's thresholding method on the cropped image to remove the background
3. computing the center of mass of the cropped image
4. cropping to a bounding cube with the computed center of mass at its center
5. extracting only the largest contour to clear the cropped image from any remaining objects that the Otsu's method failed to remove

Follows a more specific description of each preprocessing step along with the motivation behind it.

Cropping to a bounding box

The hand pose estimation stage receives from the detector a depth image with 416×416 resolution and a bounding box enclosing the hand. A 2D crop then extracts the hand from this image. This type of crop uses a bounding box for outlining the boundary for the extraction from the image plane. It does not take into account the pixels' depth values as the 3D crop does.

Otsu's thresholding

Captured images in real conditions usually contain a background, which, in most cases, is the human body, but it can also be an object on the desk. We can take advantage of the depth information, assuming the background is further from the camera. Otsu's image thresholding method [44] solves this by dividing the image pixels into two classes—foreground and background. It achieves a good performance if the histogram possesses a bimodal distribution with a valley between two peaks. Figure 5.2 shows a histogram of a depth image. The left peak closer to the camera is the hand, and the right peak is the background, which is removed by finding the valley between the two peaks. The image before and after applying thresholding is shown in Fig. 5.3.

Determining the center of mass

The idea of refinement of the detected region using the center of mass is inspired by the preprocessing of DeepPrior [43].

The function $g(a, p_{uv})$ is introduced for the clarity of the upcoming equations. The function returns zero if the pixel's value p_{uv} is zero and value a otherwise:

$$g(a, p_{uv}) = \begin{cases} 0, & \text{if } p_{uv} = 0 \\ a, & \text{otherwise} \end{cases}. \tag{5.1}$$

Then, the center of mass $\mathbf{C} = (C_u, C_v, C_z)$ is defined as a point in the depth image such that the UV coordinates are the averaged coordinates of nonzero pixels, and the Z coordinate is the average depth value of nonzero pixels:

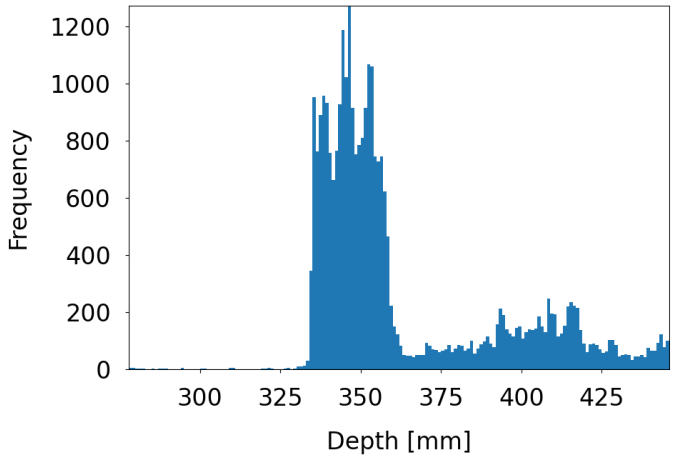


Figure 5.2: Otsu's thresholding method divides the histogram of a depth image, such as the one shown in the figure above, into foreground and background by finding the valley between two peaks. For the histogram above, Otsu's method computed the threshold to be at 378 mm.

$$
\begin{aligned}
C_u &= \frac{1}{M}\sum_{u=0}^{W}\sum_{v=0}^{H} g(u, p_{uv}) \\
C_v &= \frac{1}{M}\sum_{u=0}^{W}\sum_{v=0}^{H} g(v, p_{uv}), \\
C_z &= \frac{1}{M}\sum_{u=0}^{W}\sum_{v=0}^{H} p_{uv}
\end{aligned}
\tag{5.2}
$$

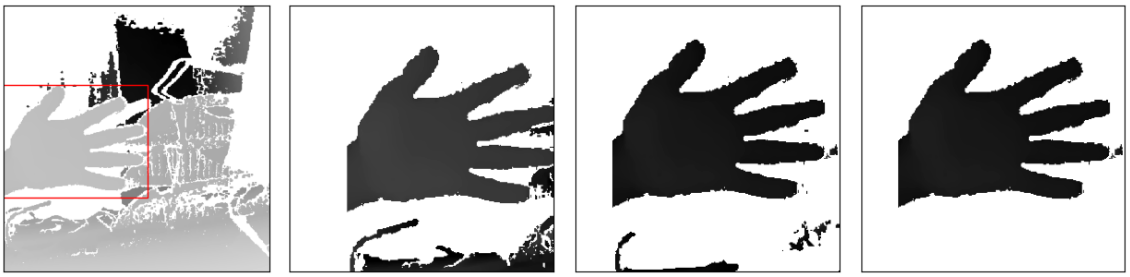
where W, H, M denote the width, height, and the number of nonzero pixels in the image.

The method tends to shift the hand from the center of the image to the edge because the computed center of mass is generally closer to the forearm rather than the tip of the fingers. On the other hand, it has the ability to slightly correct the predicted bounding box, which makes it useful for the output of object detectors.

Determining the center of image

Determining the center of the image is an alternative method to the center of mass. The center of the image approach assumes a correct position of the bounding box, requiring an effective detector or ground-truth annotations. Hence, it is suitable for training pose estimators from a dataset with known ground-truth annotations.

With the symbols from the previous section, the center of the image $\mathbf{C} = (C_u, C_v, C_z)$ is defined as the center point of the image plane with an average depth of nonzero pixels:



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

Figure 5.3: The figure shows multiple hands in various stages of preprocessing before pose estimation. The original image was captured with background objects lying on the desk. **(a)** shows a detected hand in the original image. In **(b)**, the image is cropped using a bounding cube. In **(c)**, Otsu's thresholding method was applied before computing the center of mass and also after cropping to the bounding cube. The thresholding results in a better-centered hand with less background noise. **(d)** demonstrates the resulting hand after applying all preprocessing steps, including finding the largest contour.

$$\begin{aligned} C_u &= \frac{W}{2} \\ C_v &= \frac{H}{2} \\ C_z &= \frac{1}{M}\sum_{u=0}^{W}\sum_{v=0}^{H} p_{uv} \end{aligned} \quad . \tag{5.3}$$

Cropping to a bounding cube

The bounding cube $\mathbf{B} = (\mathbf{P}_1, \mathbf{P}_2)$ with a side length a is defined by two 3D points $\mathbf{P}_1$ and $\mathbf{P}_2$ in world coordinates denoting two opposite corners of the cube, which is set around the center of mass $\mathbf{C} = (x, y, z)$ using Equation (5.4). Figure 5.4 shows the bounding cube projected onto the image plane.

$$\begin{aligned} \mathbf{P}_1 &= \mathbf{C} - \frac{a}{2} \\ \mathbf{P}_2 &= \mathbf{C} + \frac{a}{2} \end{aligned} \tag{5.4}$$

This work implements a different representation of the box, which is more convenient to work with. This representation comprises of two points $\mathbf{P}_1 = (u_1, v_1)$ and $\mathbf{P}_2 = (u_2, v_2)$ in image plane coordinates and a range z_{min}–z_{max} for the depth value (Z coordinate). The following text shows how this representation is obtained from the center of mass.

First, the center of mass $\mathbf{C} = (x, y, z)$ is converted to pixel coordinates, thus introducing $\mathbf{C} = (u, v, z)$. Next, two points $\mathbf{P}_1^w$ and $\mathbf{P}_2^w$ in the world coordinates are produced by adding or subtracting half of the side length from the X and Y coordinates while keeping the Z coordinate of the center of mass unchanged:

$$\begin{aligned} \mathbf{P}_1^w &= (x - \frac{a}{2}, y - \frac{a}{2}, z) \\ \mathbf{P}_2^w &= (x + \frac{a}{2}, y + \frac{a}{2}, z) \end{aligned} \quad , \tag{5.5}$$

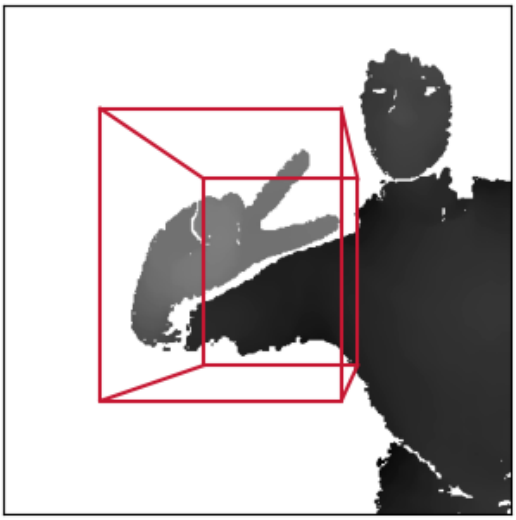


Figure 5.4: A bounding cube is created around the hand as part of the preprocessing. In the figure, the bounding cube is projected onto the image plane.

where a is the side length. These points are then converted into pixel coordinates P_1 and P_2. The Z coordinate range is retrieved by adding and subtracting half of the side length from the Z coordinate of the center of mass.

$$\begin{aligned} z_{min} &= z - \frac{a}{2} \\ z_{max} &= z + \frac{a}{2}. \end{aligned} \tag{5.6}$$

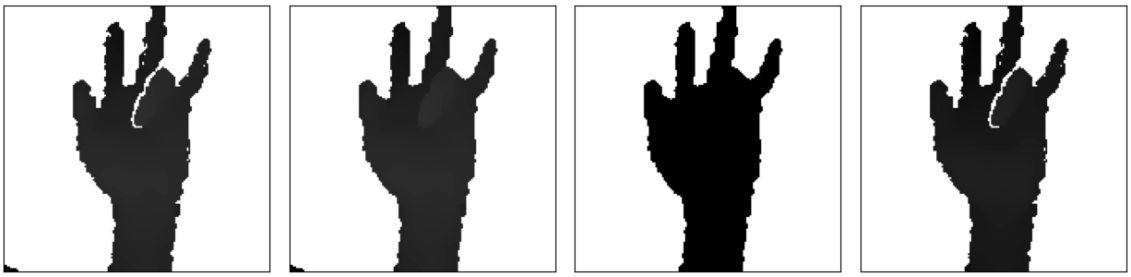
Such a representation is convenient because it allows to crop the image using the traditional 2D crop operation and then replace out-of-range depth values with zeros.

The performance of the pose estimator on the MSRA dataset improved six times by applying this normalization instead of normalizing the whole depth range. The model benefits from centering the hand inside the box and normalizing the depth to the [0, 1] range. Thus, making the prediction invariant to the distance from the camera.

The Otsu's method may seem redundant for a distant background because the bounding cube would otherwise remove it. However, to create the bounding cube, we must first find the center of mass, which would be less precisely determined if Otsu's method would not be applied because the center of mass would be computed from the background as well. This reasoning concludes that applying Otsu's thresholding method makes sense even for a distant background.

Finding largest contour

The image, after the preprocessing mentioned above, may still contain noise. Only the largest contour in the image is retained to remove the rest of the noise. First, a morphological closing is applied to remove small spaces between the fingers caused by the camera's capturing technology. Then all contours are extracted and only the largest one is used to mask the original image. The process is illustrated in Figure 5.5.



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

Figure 5.5: All remaining noise can be removed by finding the largest contour. Image **(a)** shows the image before this preprocessing step is applied. Notice the black spot in the bottom left corner. First, a morphological closing is applied on the image, which produces **(b)**. The white space between fingers is removed, which is essential because the largest contour in some poses would not otherwise include some fingers that are not directly connected. Finally, masking the original image with the largest contour **(c)** produces **(d)**, removing the spot in the bottom left corner.

5.2.4 Gesture acceptance

The gesture acceptance module performs gesture recognition and provides feedback on the predicted pose. The feedback says which joints are incorrectly placed to satisfy a registered gesture or an error in the hand's orientation. The module receives a skeleton representation of the hand from the hand pose estimation stage and calculates the similarity between the received skeleton and the precaptured hand poses stored in a local database. The gesture is accepted if the comparison yields errors below specified thresholds.

Both the hand shape and orientation in space constitute a gesture. Consequently, the similarity between poses is given by two aspects. The first one is the joint relation error metric that defines the similarity in terms of hand shape. The second one is the difference between the hands' orientations given by an angle between the two poses computed from the joints delimiting the hand's palm.

Joint relation error

Section 2.3 discussed solutions for gesture recognition. They train parametric classification models (e.g., FGMM, SVM). The proposed method, which was largely motivated by the need for feedback, differs from these, as it is a nonparametric classification method. It can be viewed as k-nearest neighbors algorithm, where $k = 1$. It uses features called joint relation errors. In other words, the gesture is classified as the one with the lowest sum of mean joint errors. The extraction of these features is described below.

Figure 5.6 illustrates the computed joint relation errors on two distinct gestures. A joint is represented by a red dot whose color intensity and size are proportional to the joint relation error. The index and middle fingers show lower joint relation errors because they are in the same position in both gestures. Furthermore, each hand has a yellow arrow indicating its orientation.

Let $\mathbf{Z}_k = (x_k, y_k, z_k)$ be the absolute position of the k-th joint in world coordinates. Then, to ensure the joint features are invariant to shifting, the joints are subtracted from each other, transforming into relative positions:

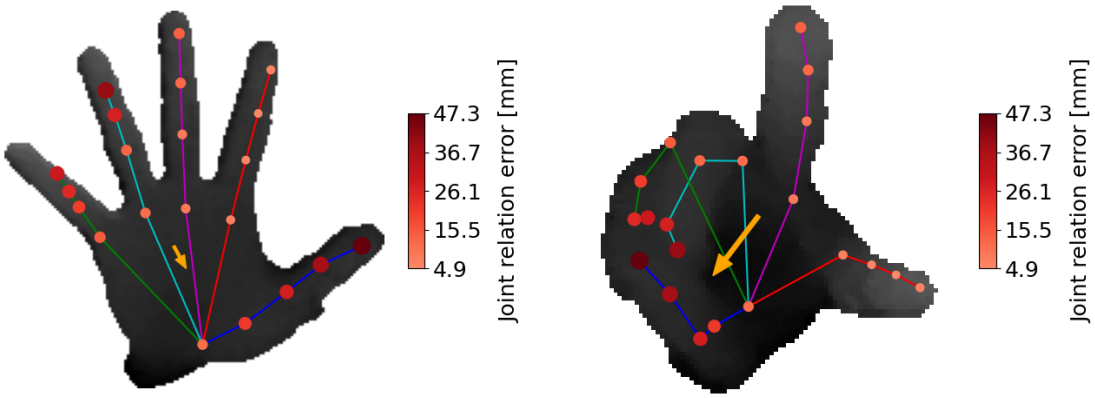


Figure 5.6: The figure shows the difference between two gestures from the MSRA dataset using the joint relation errors. A joint is represented by a red dot whose size is proportional to the joint error. Yellow arrows indicate the hands' orientations.

$$\mathbf{R}_{mn} = \mathbf{Z}_m - \mathbf{Z}_n, m, n \in \{1, \ldots, N\}, \tag{5.7}$$

where N is the number of joints. This step is similar to computing the relative positions from Section 2.3, where only the wrist joint was used to compute the relative distances. This work computes the relative positions for each joint, producing an array of N^2 relative positions as opposed to N. It is motivated by the need to preserve the relative joint locations because the information would be lost in the next step:

$$R_{mn} = \|\mathbf{R}_{mn}\|_2, m, n \in \{1, \ldots, N\}, \tag{5.8}$$

where $\|\ldots\|_2$ denotes an ℓ^2 norm.

To ensure invariance to scaling, needed mainly because hands have different sizes, the relative distances are scaled by a certain factor, normalizing hands to the same size. First, the length of each finger L_t is computed as the sum of Euclidean distances between MCP, DIP, PIP, and TIP joints' locations:

$$L_t = \|\mathrm{MCP}_t - \mathrm{PIP}_t\|_2 + \|\mathrm{PIP}_t - \mathrm{DIP}_t\|_2 + \|\mathrm{DIP}_t - \mathrm{TIP}_t\|_2, \tag{5.9}$$

where $t \in \{1, \ldots, 5\}$ denotes a finger's index. The mean length of a finger M is produced by averaging the fingers' lengths:

$$M = \frac{1}{5} \sum_{i=1}^{5} L_i. \tag{5.10}$$

A standard mean length is defined to scale hands to the average hand size. The standard mean length was determined to be 65 mm from the MSRA dataset. Then, it is divided by the mean length of the five fingers, producing a scale factor s:

$$s = \frac{65}{M}. \tag{5.11}$$

The relative distances are scaled according to the computed scale factor:

$$R_{mn} = R_{mn} * s. \tag{5.12}$$

Then, the average difference of relative distances between the k-th joint and all other joints gives the joint relation error score E_k:

$$E_k = \frac{1}{N-1}\sum_{i=0}^{N}|R^1_{ki} - R^2_{ki}|, k \in \{1, \ldots, N\}, \tag{5.13}$$

where R^1_{ki} and R^2_{ki} denote the relative distances between the k-th and i-th joint of two hands, respectively. The sum of all E_k components gives the final joints relation error (JRE) score.

Orientation difference

The difference in orientation of the input pose and the registered pose is computed as the angle between the normal vectors of the hyperplanes that best fit the palms' joints. Six joints enclose the palm—the MCPs and a wrist joint (see the skeletal hand model in Section 2.3 for reference). The thumb's MCP is not included in the hyperplane approximation because it is too flexible and could distort the fitted hyperplane by placing the thumb in a direction that does not conform to the palm.

The best-fitting hyperplane through the palm joints can be approximated through Singular Value Decomposition. Let $a = [p_1, \ldots, p_5]^T$ be a set of 3D points. First, the points are normalized by subtracting their mean $c = \frac{1}{5}\sum_{i=1}^{n} p_i$ introducing matrix $M = [p_1 - c, \ldots, p_5 - c]^T$. The Singular Value Decomposition of $M \in \mathbb{R}^{5x3}$ is $M = U\Sigma V^T$, where $U \in \mathbb{R}^{5x5}$ and $V \in \mathbb{R}^{3x3}$ are orthogonal matrices. Columns of U span the column space of M (collections of X, Y, and Z coordinates), and columns of V span the row space of M (the 3D points). Because all points should lie in a plane, the minimal basis that spans it is the first two columns of V. The third column of V will not span the plane but be normal because V is orthogonal. [6]

The angle representing the orientation difference is obtained with an arccosine of the dot product of the normal vectors a and b:

$$\theta = \arccos(\frac{a \cdot b}{\|a\|\|b\|}). \tag{5.14}$$

The current system's design does not support the differentiation between the left and right hands. For practical reasons, it assumes that one of the hands' palms always faces the camera. This assumption implies that the difference in orientation never exceeds 90 degrees. Hence, the computed angle, as per Equation 5.14, is corrected:

$$\theta = 180 - \theta, \tag{5.15}$$

transforming the angles into the [0, 90] range.

Feedback

The user receives feedback in the form of images showing how much the joints' positions differ from the expected gesture, as shown in Figure 5.6. They also get a label of the most similar gesture as the output of the gesture recognition. This information is beneficial if more gestures are registered. The system is also able to tell the difference in orientation.

5.2.5 System's limitations

One should bear in mind that the designed system has some limitations that must be considered before use. The performance of gesture recognition may be affected by a fast motion of the hand, hand occlusion, or a partially out-of-view hand. The differentiation of the left and right hands is also outside the capability of this system.

The hand's fast motion may cause a slightly distorted image that could worsen the pose estimation and result in inaccurate gesture classification. However, it has not been empirically proved to make a noticeable difference.

Occluded or partially out-of–of-view hands are challenging tasks that have to be specifically tackled in hand pose estimation; otherwise, the inferred pose will be incorrect. This work does not take these into account, and as a result, these are considered the system's limitations.

The inability to differentiate between hands is due to the nature of the skeletal hand representation. It might be possible to reason about this differentiation for certain poses based on the thumb's position and the way the fingers are bent. However, it is almost impossible for a gesture with stretched fingers to tell whether the skeletal information represents the left hand or the right hand rotated by 180 degrees along the wrist. In that case, it might be helpful to look at the human pose as a whole to determine which hand it is, which could be solved during the detection stage. Still, it would probably fail if the human body is outside the image.

Chapter 6

Implementation and training

This chapter gives a brief overview of the tools used for implementation, including libraries and computational resources. It is followed by sections presenting the actual implementation of the system and the training of models.

6.1 Toolset

The system was implemented in Python. It was chosen for its popularity among the scientific community and the availability of libraries for machine learning, such as Tensorflow. I have used this library several times and had no reason to use any alternatives such as Theano and PyTorch, which are also favored by the scientific community.

Tensorflow[1] is a machine learning library developed by Google. Machine learning scientists use it for various tasks; however, its primary focus is deep neural networks. A few years ago, Keras library was made a part of Tensorflow. The Keras library is a powerful and high-level library for building, training, and evaluating deep neural networks.

As for the other libraries, the most notable is NumPy. It is a library adding support for working with multidimensional arrays, also called tensors. [22] It also provides high-level mathematical functions to operate on these arrays. Many libraries are based on NumPy, with no exception of Tensorflow.

I also used Pillow[2] library, which is an imaging library in Python, providing many functions for processing images, such as reading, resizing, rotating, and many other transformations. It was primarily used for reading images from a file system.

The scikit-learn[3] and scikit-image[4] are libraries providing tools for machine learning and image processing. They were used for evaluation metrics and Otsu's thresholding.

OpenCV[5] is used to extract contours from the depth image as a part of preprocessing. It provides many image processing functions, including morphology operations.

Pyrealsense2[6] is used to communicate with an Intel RealSense camera and capture live depth images. It is a wrapper for Intel RealSense SDK 2.0, providing C++ to Python bindings. The library allows countless operations within the limits of the camera in use, such as: capturing both color and depth images, applying filters to these images, reading the

[1] http://tensorflow.org
[2] https://pillow.readthedocs.io
[3] https://scikit-learn.org
[4] https://scikit-image.org
[5] https://opencv.org
[6] https://pypi.org/project/pyrealsense2

camera configuration, and setting depth units and maximum distance for a depth sensor. Each camera is different and allows only certain operations. For example, the SR305 camera does not support changing the depth unit. Hence, the values have to be amended by the user.

6.1.1 Metacentrum

Apart from my computer, which is highly inefficient for training deep models, I used a service called Metacentrum[7]. This service is provided by the MetaCentrum Virtual Organization, which also runs cloud services and a Hadoop framework. Metacentrum offers resources for grid computing. Such an infrastructure provides resources consisting of interconnected computers. The users submit their jobs on frontends, and when the resources are available, they are run on computational nodes. A scheduling system tracks the grid's available and used resources and schedules jobs from a queue to run on computational nodes. The users can define the required resources, such as GPU, CPU, and memory, when submitting their job to the queue. The scheduling system schedules the job once the resources are available.

6.2 Recognition system

The designed system from Section 5.2 was implemented in Python, whose main components are displayed in a class diagram in Figure 6.1. During instantiation, the instance of `HandPositionEstimator` loads the saved detection and estimation models. Additionally, the registered poses are read from a local database representing the target gestures. The system starts by calling `start` method of the `GestureAccepter` class and providing it with an image generator. It iterates over the generator, yielding either live imagery or images from a disk. It calls `infer_from_image` method of the `HandPositionEstimator` class to determine the hand's skeleton. The two models are used to predict the joints' locations by first detecting the hand using the Tiny YOLOv3 detector, then extracting the image's patch as defined by the bounding box and preprocessing for pose estimation. It is then passed to a JGR-P2O model, which determines the precise joints' locations. Then, it is postprocessed, and the result is returned to `GestureAccepter`. Lastly, the `start` method calls the `accept_gesture`, which compares the determined pose with the database of registered poses, using the mechanism introduced in Section 5.2.4. The method returns `GestureAcceptanceResult` containing all information regarding the comparison. Based on this information, the `start` method returns the gesture's label if the input pose turns out to be one of the registered gestures.

Tiny YOLOv3 and JGR-P2O were both implemented in Tensorflow according to papers in which they were presented. However, some parts of the YOLOv3 network's architecture were inspired by the implementation of YunYang1994 [67], specifically the loss function, YOLO layer, and preparation of data.

6.2.1 Live image generator

The `GestureAccepter` expects an image generator as a parameter to the `start` method. It iterates over the generator and checks if it is the desired gesture for each image. The system is aimed to be used in real-time using real imagery. The pyrealsense library allows,

[7] https://wiki.metacentrum.cz/

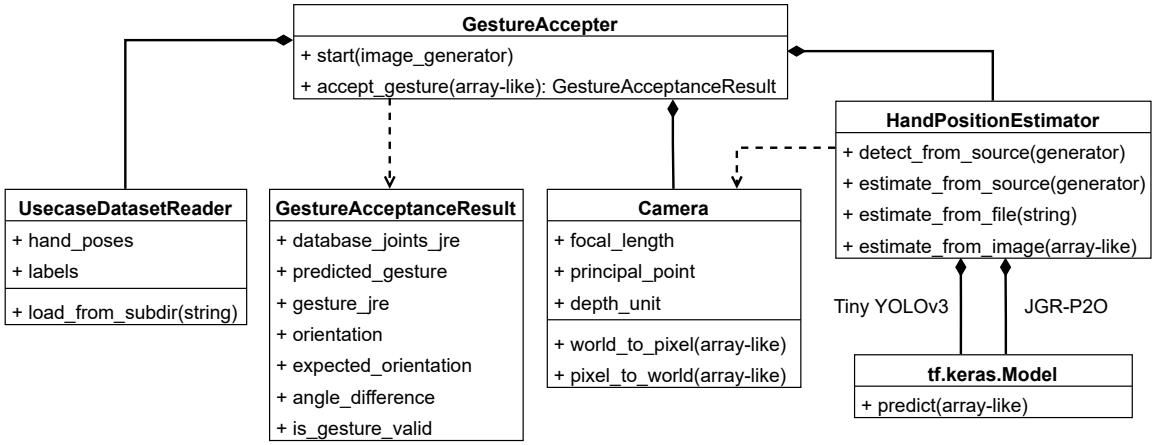


Figure 6.1: The proposed gesture recognition system is composed of several classes whose diagram is shown in the figure. `UsecaseDatasetReader` provides access to registered poses the system recognizes. `HandPositionEstimator` encapsulates determination of hand's skeleton, which is compared to the registered poses in the `GestureAccepter`, returning `GestureAcceptanceResult` containing information about the comparison. `Camera` represents the camera's intrinsic parameters in use and provides methods for conversion between pixel and world coordinates.

among other operations, reading images from Intel RealSense cameras. The following listing demonstrates a generator yielding captured images.

```
import pyrealsense2 as rs

def generate_live_images():
    # Get a~RealSense pipeline
    pipe = rs.pipeline()
    # Configure the depth stream for the pipeline
    cfg = rs.config()
    cfg.enable_stream(rs.stream.depth, 640, 480)
    # Start the pipeline with the configuration
    pipe.start(cfg)
    try:
        while True:
            # Receive frames
            frameset = pipe.wait_for_frames()
            # Get the depth information from the frame
            depth_frame = frameset.get_depth_frame()
            # Convert the depth data to numpy array
            depth_image = np.array(depth_frame.get_data())
            # Add a~single channel dimension, producing shape (480, 640, 1)
            depth_image = depth_image[..., np.newaxis]
            yield depth_image
    finally:
        pipe.stop()
```

Hand pose estimation preprocessing

The preprocessing applied before hand pose estimation, as described in Section 5.3, was implemented in NumPy and Tensorflow together with scikit-image and OpenCV libraries. The threshold that divides the image into foreground and background is retrieved by calling the `threshold_otsu` function from the scikit-image library. The rest of the noise is removed by extracting the largest contour. In order to do so, morphological closing is applied to the image, which is implemented by calling OpenCV's `morphologyEx` function with a structuring element of shape 5×5. Then, all contours are extracted using the `findContours` function. The largest one is used to mask the original image.

Gesture acceptance module

The equations from Section 5.2.4, regarding relative positions (5.7) and relative distances (5.8), are implemented in a vectorized form in NumPy. First, a distance matrix $\mathbf{R}$ is computed for each hand. This symmetric matrix contains all relative distances between each pair of joints. Then, both matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ are subtracted from each other, and the sum over each row of the absolute values represent the Joint Relation Errors, producing a vector $\mathbf{E}$ of 21 scalars, one for each joint. This vectorized approach computes the joint relation errors for each pair of hands between the input pose, and all registered poses simultaneously, which significantly improves the algorithm's efficiency.

The hand's orientation is represented by a vector normal to the hyperplane that best fits certain joints, as described in the previous chapter. First, the joints' positions are normalized by their means to normalize their coordinates. After that, the `np.linalg.svd` function from NumPy library computes singular value decomposition, where the normal vector is the last row of the `vh` matrix, as demonstrated in the listing below.

```
x -= np.mean(x, axis=0)
u, s, vh = np.linalg.svd(x)
normal_vector = vh[-1]
```

6.3 Training

Both models from Chapter 3 were trained in Metacentrum in Adan cluster[8] on a single GPU. The implementation of the training pipeline is discussed in the next section, followed by a more specific description of the training of those models.

6.3.1 Training pipeline

Both the detection model and the estimator were trained on large datasets with sizes of tens of gigabytes. Figure 6.2 shows a general training pipeline. A compiled model trains on a dataset provided by a dataset pipeline that prepares data for the model. The training process outputs logs after each batch or epoch and saves the trained model for later use.

Dataset pipeline

The main pipeline is set up using a `tf.data.Dataset`, followed by a preprocessor modifying the images and annotations into a suitable form for the specific model. The Tensorflow

[8] https://wiki.metacentrum.cz/wiki/Cluster_Adan

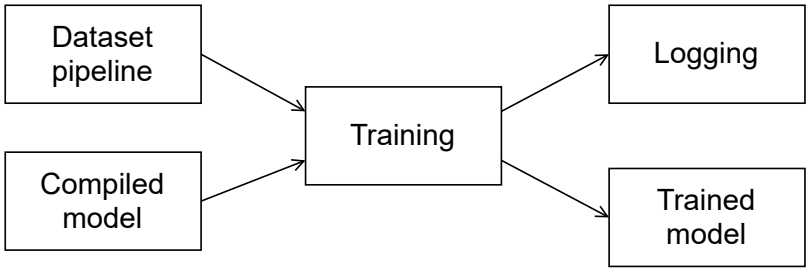


Figure 6.2: The figure depicts a general training process used for training an object detector and hand pose estimator. The training requires a compiled model and a dataset pipeline providing preprocessed data, and produces logs and the trained model.

dataset pipeline provides a way for working with large datasets that cannot be loaded into memory all at once. Instead, it prepares only the next batch. Generally, the pipeline consists of creating the source dataset from the input data, applying transformations to the data, and, finally, iterating over the dataset in a batch-wise way.

The Tensorflow's `tf.data.Dataset` class allows building an asynchronous, highly efficient pipeline. It loads data from a disk in batches and applies optimized transformations. The specific pipeline in the listing below loads the HandSeg dataset. First, the `tf.data.Dataset` is instantiated from annotation lines using `from_tensor_slices` function. All annotations are shuffled after each iteration using `shuffle` function. he `repeat` function is called to repeat the dataset indefinitely to make sure that the pipeline never runs out of data. Each annotation line contains a file name and ground-truth bounding box annotations. The image is loaded in the custom `_prepare_sample` function registered with `map`. Then, a batch of a certain size is created and padded with zeros using the `padded_batch`, which is required to allow images to contain a different number of objects. The `prefetch` function allows other elements to be loaded while the current batch is being processed. Pipelines for other datasets are built similarly.

```
dataset = tf.data.Dataset.from_tensor_slices(annotations)
dataset = dataset.shuffle(buffer_size=len(annotations),
                          reshuffle_each_iteration=True)
dataset = dataset.repeat()
dataset = dataset.map(_prepare_sample)
shapes = (tf.TensorShape([416, 416, 1]), tf.TensorShape([None, 4]))
dataset = dataset.padded_batch(batch_size, padded_shapes=shapes)
dataset = dataset.prefetch(buffer_size=1)
```

Further preprocessing is performed after the Tensorflow pipeline using a Python generator. Fortunately, the Keras API also supports python's generators, providing a way to modify data in an intermediate place. The general structure of the generator is depicted in the listing below. Although it provides certain flexibility for reusing the Tensorflow pipeline for different models, it would be more efficient to incorporate the preprocessing from the generator in the Tensorflow pipeline instead.

```
class DatasetPreprocessor:

    def __init__(self, dataset):
```

```
        self.dataset = dataset

    def __iter__(self):
        return self

    def __next__(self):
        batch_data = self.dataset.get_next()
        preprocessed_data = self.preprocess(batch_data)
        return preprocessed_data

    def preprocess(self, data):
        """ Preprocessing """
```

Building a model

The models are built as instances of `tf.keras.Model` class, and composed of layers from the `tf.keras.layers` package, e.g., `Input`, `Conv2D`, `ReLU`, `MaxPooling2D`. For example, the JGR-P2O model is instantiated as follows:

```
model = Model(inputs=input_layer, outputs=[uvz, offsets])
```

where `input_layer` is an instance of the `Input` layer, and `uvz` and `offsets` are references to nodes in the Tensorflow's computational graph, producing the UVZ coordinates and offsets.

The model is then compiled using the `tf.Model.compile` function, providing it with an optimizer, loss functions, and metrics. It does not reset the model's weights if any were loaded prior to calling this function. It sets the mentioned parameters which are required for training. The two models use different loss functions and metrics, but both of them use the Adam optimizer.

Training

Training starts by calling `fit` function on the compiled model. The function expects a dataset and other parameters, such as the number of epochs and callbacks. The callbacks are especially useful for executing actions during training. Several of them are available in the `tf.keras.callbacks` package. The models were trained with the following callbacks:

TensorBoard
: During training, the TensorBoard callback provides logging of loss functions, metrics, and learning rate. The logged information can be displayed in real-time in TensorBoard. Since version 2.3, it supports reading the data logs into a Pandas DataFrame for performing post hoc analysis and creating custom visualizations—Figure 6.3 is an example of this approach.

ModelCheckpoint
: The ModelCheckpoint saves the model or its weights periodically after each epoch or after a specified number of batches. It also allows specifying which loss function to monitor and whether to save only the best model.

EarlyStopping
: The EarlyStopping is handy for stopping the model from overfitting. Once the monitored loss stops improving, it ceases training.

TerminateOnNaN

This particular callback is helpful during the development of the model. The training is stopped if NaN loss is encountered.

Trained model

At the end of the training, the model's weights are saved using `save_weights` function. These can be later used to initialize the model with `load_weights`. The model's weights are also saved during training by the ModelCheckpoint callback.

6.3.2 Training of Tiny YOLOv3 detector

Tensorflow, together with its Keras API, was used for building the network, as presented in Section 6.3.1. The model's architecture is loaded from a config file that was downloaded from the original source[9], and the number of filters of two convolutional layers was modified from 255 to 15 to accept a single class instead of the original 80. Adam optimizer was used for training with a decay rate of 0.94 each epoch.

The model was trained for 12 epochs on the HandSeg dataset introduced in Section 4.2.1. The training took six hours on NVIDIA Tesla T4 16GB and used about 25 GB of memory. The validation loss during training was getting lower with each epoch. Generally, as long as the validation loss is decreasing, the model is improving. [21] The plot of the training and validation losses, Fig. 6.3, shows that while the training loss was still decreasing, the validation loss was not. Hence, the training stopped after the fourteenth epoch to avoid overfitting.

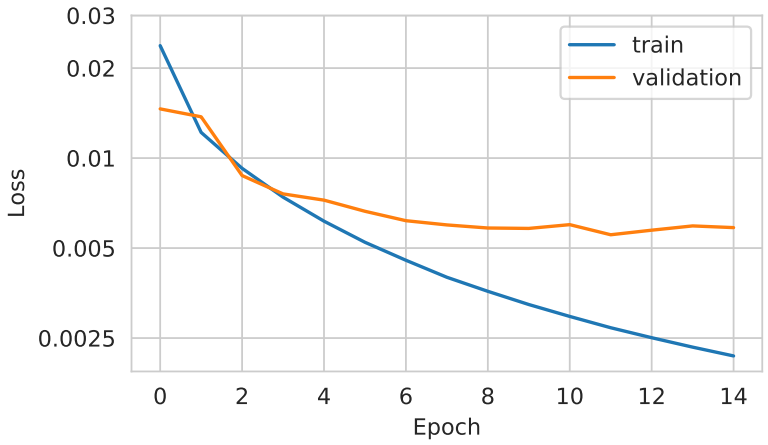


Figure 6.3: TinyYOLOv3 was trained to detect hands from depth images using the HandSeg dataset. The figure shows training and validation losses during the training.

6.3.3 Training of JGR-P2O estimator

The JGR-P2O architecture, as described in Section 3.2, was implemented in TensorFlow and trained on MSRA and BigHand datasets. The authors of the method stacked two hourglass networks to improve the accuracy slightly. However, more hourglasses increase

[9] https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-tiny.cfg

the architecture's complexity, decreasing its efficiency. They showed that a single hourglass could achieve 8.62 MJE on the NYU dataset, while two hourglasses achieve 8.29, which is only a slight improvement, considering the architecture's complexity is almost doubled. Thus, this work uses only a single hourglass.

The model was trained on the MSRA and BigHand datasets in Metacentrum on nodes with an NVIDIA Tesla T4 GPU. The input image is preprocessed and resized to a fixed size 96×96 using a resizing method that combines bilinear and nearest-neighbor interpolation. The bilinear interpolation on its own would produce invalid values for neighboring pixels of the background (zero pixels) and foreground (non-zero pixels). The nearest-neighbor interpolation comes into play to resolve the border between the foreground and background pixels by choosing the nearest value instead of interpolating the value with a linear function. The depth values are normalized to the [0, 1] range. For training, the batch size was initially set to 32, and later, for subsequent training, it was increased to 64. Adam optimizer with an initial learning rate of 0.0001 is used with a decay rate of 0.96 after every 512 batches.

The BigHand dataset contains low variation in in-plane rotation. Therefore, the dataset is augmented online during training, similar to how DeepPrior [43] performs augmentation, including 3D scaling, 3D translation, and in-plane rotation. The scaling and translation are performed on the bounding cube. Scaling shrinks or enlarges the size of the box with a normal distribution $N(\mu = 1, \sigma^2 = 0.08)$ to ensure invariance to scaling. The translation moves the bounding cube in world coordinates in relation to the center of mass inside the box by a value given by a normal distribution $N(\mu = 0, \sigma^2 = 8)$ in millimeters. The image is rotated in-plane with a uniform distribution $U(-180, 180)$ in degrees.

Up to six subjects were used from the BigHand dataset, each consisting of 260 thousand samples. The side of the bounding cube is set to 20 cm. The training took about three days on average. It was observed that a high learning rate was preventing the model from improving, which manifested in fluctuating performance between epochs—the predicted skeleton was not regressing to a single location but was jumping around the image, which was solved by manually decreasing the learning rate.

Chapter 7

Evaluation and testing

Each part of the dataset—hand detection, hand pose estimation, and gesture recognition—was separately evaluated. Additionally, the system as a whole was evaluated on the custom dataset, as well as tested in real-time. The following section introduces the evaluation metrics for binary classifiers.

7.1 Evaluation metrics for binary classifiers

Although the most basic metric for classification is accuracy, it is usually not the best choice, especially in an imbalanced classification problem. In a binary classification problem, the accuracy metric does not provide the necessary information about the system. For this reason, other evaluation metrics are introduced to analyze the quality of classifiers.

Metrics for binary classification can be expressed using terminology from a confusion matrix [12]. This matrix can be created for a classification problem of arbitrary size. This work is concerned only with binary classification; therefore, the discussion will focus on a two-class confusion matrix shown in Table 7.1.

		True class	
		P	N
Predicted class	$\hat{P}$	True Positive	False Positive
	$\hat{N}$	False Negative	True Negative

Table 7.1: Düntsch, I. and Gediga, G. presented a 2-class confusion matrix. [12]

The confusion matrix divides the predictions into four sets. The true class of a prediction can be either positive or negative. The ideal state is when no false negatives and false positives are present, which means both classes were correctly predicted.

The most common metric is accuracy and is defined as the number of correct predictions divided by the total number of predictions.

$$\text{accuracy} = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \tag{7.1}$$

In an imbalanced classification problem, the most commonly used metrics are precision and recall. Precision describes how many of the predictions as a positive class indeed belong to the positive class. Recall says how well the positive class was predicted.

$$\text{precision} = \frac{T_p}{T_p + F_p} \tag{7.2}$$

$$\text{recall} = \frac{T_p}{T_p + F_n} \tag{7.3}$$

Another useful metric is F_β score that combines both precision and recall:

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \tag{7.4}$$

The more important the recall is over precision, the higher the β. The precision is more important with $\beta \in (0, 1)$. On the other hand, with $\beta > 1$, the recall has more weight in the F_β score. For example, the F_1 score expresses a balance between precision and recall, meaning they are equally important. [10]

The precision-recall curve

The precision-recall curve can visualize a tradeoff between precision and recall. Precision and recall are calculated for every threshold and drawn on a plot. Generally, the higher the precision, the lower the recall. With the precision-recall curve information, we can choose the best precision-recall tradeoff, usually right before the recall starts to drop rapidly. [21]

Intersection over union

Intersection over union [51] (IoU) metric is useful especially for the evaluation of object localization in object detection. The area of intersection of the ground truth bounding box A and the predicted bounding box B is divided by the area of their union:

$$\text{IoU} = \frac{|a \cap B|}{|a \cup B|}. \tag{7.5}$$

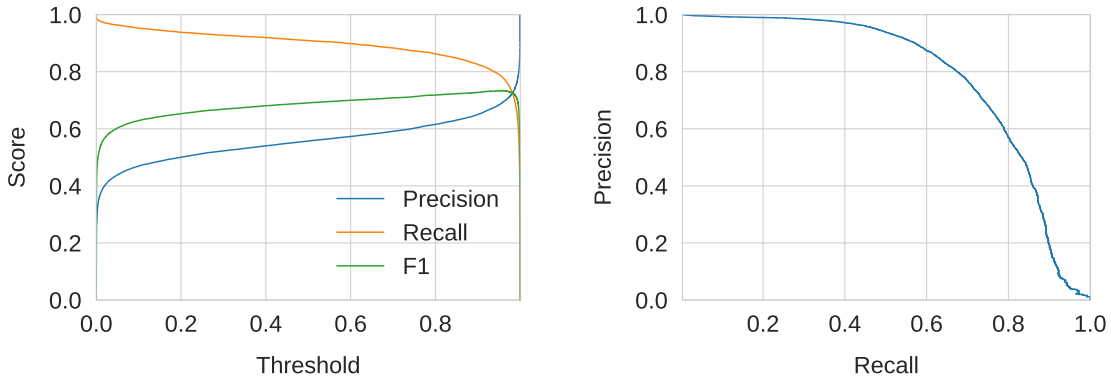
The IoU takes values in the range from 0 to 1. The value 1 indicates a perfect overlap, while the value 0 means no overlap.

7.2 Hand detection

Object detection consists of object localization and object classification, as described in Section 2.1. Both stages of the Tiny YOLOv3 model were evaluated on the HandSeg dataset. The model produces bounding boxes and probabilities as a result of localization and classification, respectively.

Object classification is evaluated with precision, recall, and F1 metrics, and the precision-recall curve depicts the metrics' relationship. As Figure 7.1 indicates, the model achieves the optimal balance between precision and recall with a 0.95 threshold, where the F1 score is highest.

The hands' localization was evaluated using the IoU metric, which was also used as a part of a training loss. The detector predicts at two scales, as described in Section 3.1. The IoU was calculated for all responsible boxes over both scales, achieving 86.35 % IoU on the



(a) Precision, recall, and F1 score

(b) Precision-recall curve

Figure 7.1: The Tiny YOLOv3 detector was trained and evaluated on the HandSeg dataset. Figure **(a)** shows precision, recall, and F1 evaluation metrics and **(b)** shows a precision-recall curve.

HandSeg dataset. One of the scales produces a higher IoU than the other one, depending on the hand's size in the image. Therefore, the result is slightly better in practice. Figure 7.2 shows predicted bounding boxes on the validation part of the dataset.

Even though the Tiny YOLOv3 is a simpler variation of YOLOv3, it detects hands successfully. I believe this is due to the small number of classes that the network has to differentiate.

7.2.1 Evaluation with SR305 and D415 cameras

I tested the detector with both cameras at my disposal. The main difference between them for the purposes of this work is their operating range. SR305 can capture at a distance of 0.2–1.5 meters, unlike a D415, which can capture at a distance of 0.5–10 meters. The detector was trained on the BigHand dataset, created with an SR300 camera, similar to an SR305. The model struggled to recognize hands in raw depth images captured with a D415. The camera captures values well beyond what the model saw in the dataset during training. I tried removing the background at a greater distance than 1.5 meters, and it detected hands without any difficulties. It is not the background that causes the model to struggle, but rather the values above 1.5 meters as these were not included during training.

Figure 7.3 shows predictions of the detector during real-time testing using the SR305 camera.

7.2.2 Random Decision Forest for depth-based hand detection

I also implemented a method proposed by Myoung-Kyu Sohn et al. [50] which utilizes a Random Decision Forest to detect hands in depth images. This method showed to be overly memory intensive due to feature extraction. Unfortunately, the technique did not prove to work as indicated in the paper. The evaluation discovered that it always detected the closest object regardless of whether it was a hand. For these reasons, I resorted to a more robust solution—a deep learning model.

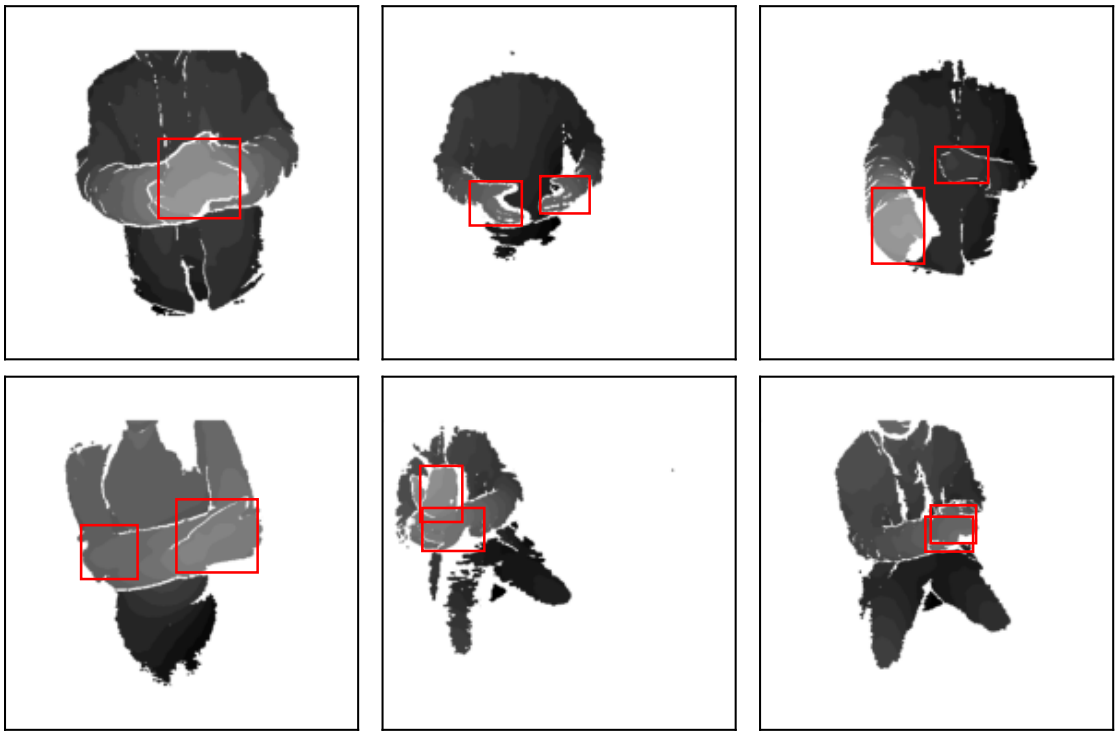


Figure 7.2: The Tiny YOLOv3 model was evaluated on the HandSeg dataset. The red bounding box indicates a detected hand. A non-max suppression [68] postprocessing method is performed on the output of the model to eliminate boxes with a small confidence and prune boxes with high IoU with previously selected boxes. The confidence threshold was selected as 0.5.

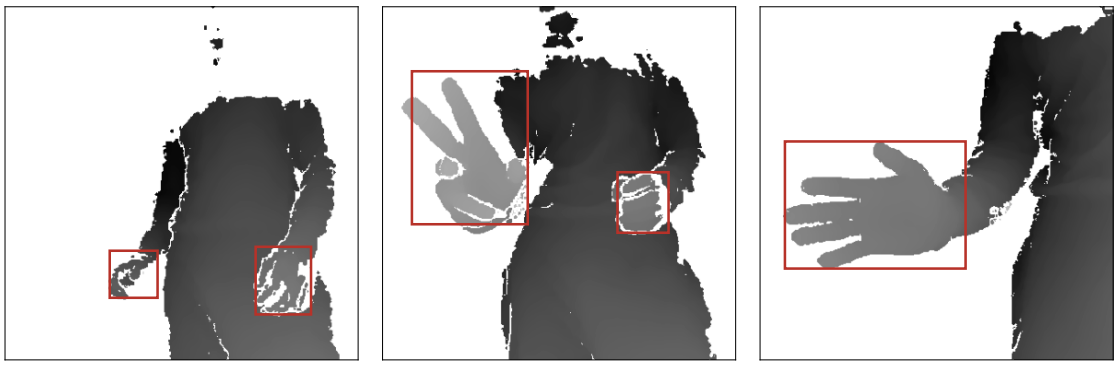
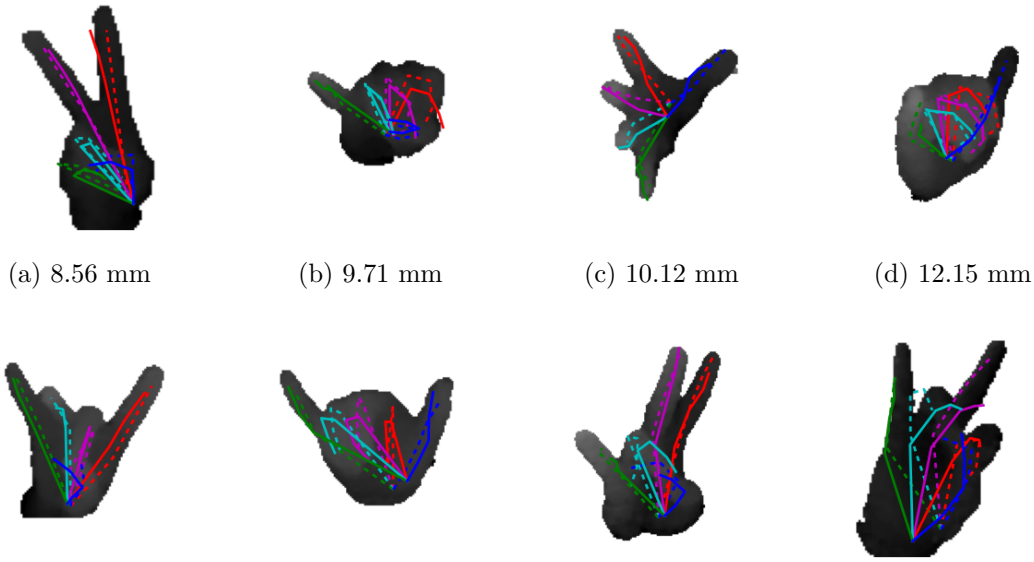


Figure 7.3: The Tiny YOLOv3 model was tested in real-time using the SR305 camera. The red bounding boxes indicate the detected hands.

7.3 Hand pose estimation

The JGR-P2O model achieves a mean joint error (MJE) of 14.7 mm after training for 53 epochs, which took two and a half days. The side length of the normalizing bounding cube was set to 180 mm, which is probably unnecessarily too big. Even though this result does not conform to what the authors of the JGR-P2O achieved after 58 epochs, the model would probably achieve better results if the learning rate was further decreased and if the side

length of the bounding cube was smaller than 180 mm. Because the hands lack forearms in the MSRA dataset, the training was focused on the BigHand instead. Figure 7.4 shows predictions on randomly selected images from the MSRA dataset.



(a) 8.56 mm (b) 9.71 mm (c) 10.12 mm (d) 12.15 mm

(e) 13.12 mm (f) 14.23 mm (g) 15.64 mm (h) 17.00 mm

Figure 7.4: The figure shows predicted joints (solid lines) with ground truth annotations (dashed lines) on randomly selected images from the test MSRA dataset. The millimeters below images indicate the mean joint error.

Figure 7.5 depicts some of the better predictions on images from the BigHand dataset. The hands were extracted using the presented hand detector and then preprocessed according to Section 5.2.3. The estimator was also tested in real-time, as shown in Figure 7.6. The estimator poorly generalizes to unseen poses.

Observing that the model struggles to estimate some of the poses motivated the increase of the training dataset's size, expecting the model to improve by training on a wider variety of viewpoints and articulations. Hence, the number of subjects was increased to six, containing 1.5 million images in total. The results demonstrated no significant improvement.

Currently, the hand pose estimator's performance is poor. Even though the results on the MSRA dataset seem promising, the results on the Bighand and the custom dataset seem to be inadequate for further application. It may be approached in several ways. Specifically, the image for the estimator could be rid of the arm, as in the MSRA dataset, another method for hand pose estimation could be selected, or a new training dataset could be captured to cover a wider variety of hand poses.

Another method for hand pose estimation could be selected from the methods that participated in HANDS17 and HANDS19 challenges. Both challenges used the BigHand dataset, which contains real-like images. They showed high performance on this dataset. For example, V2V-PoseNet is one of the best-performing methods in these challenges. The JGR-P2O paper also compared hand pose estimation methods, showing that JGR-P2O surpasses the V2V-PoseNet in all three datasets (ICVL, NYU, MSRA), implicating that the JGR-P2O should achieve results similar to the V2V-PoseNet on the BigHand dataset. However, the MSRA dataset has lower viewpoint and articulation variations than BigHand,

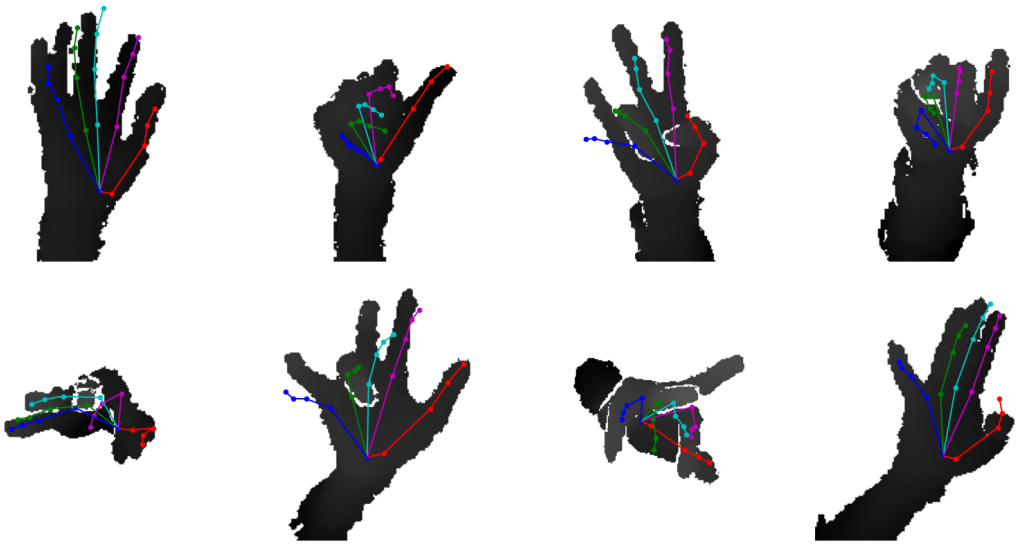


Figure 7.5: The figure shows some of the better predictions on the BigHand dataset. Although the model fails for certain poses, it demonstrates high potential, considering it was trained on images from two subjects. The side length of the bounding cube was set to 230 mm, and the center of mass was applied. The forearm causes the hand to shift to the side of the image plane.

making it easier for the estimator to achieve better results. Hence, the question remains whether the JGR-P2O's capacity is sufficient for achieving state-of-the-art results on complex datasets such as the BigHand. Therefore, an experiment was conducted to answer this question. The model was trained with an increased feature dimension (196, instead of the original 128), increasing the number of trainable parameters from 0.7 million to 1.6 million. After several days of training, the model achieved no visible improvement, concluding that the model's capacity is probably not the cause.

I believe the image preprocessing is partly the cause of those poor results, specifically the presence of the arm. As already discussed, the MSRA dataset contains no arm in contrast to the BigHand. Removing the arm would bring many benefits. First of all, the hand would be centered on the bounding cube and stretched over the whole cube. The size of the cube could be decreased to fit the hand just right, drastically improving the normalization. Instead of locating the hand with object detection, it could be located with image segmentation trained on the HandSeg dataset. The segmentation would locate pixels belonging to the hand only, effectively removing the forearm. Another benefit of this approach is that the extracted subregion containing the hand would contain much less background, possibly making the preprocessing more reliable. The issue with preprocessing was partly eliminated by setting the length of the bounding cube's side to 150 mm for training. The model improved significantly, reaching roughly 24.9 MJE. It is an acceptable result, considering the BigHand's complexity when compared to the MSRA.

The poor generalization to unseen poses is probably due to the composition of the BigHand dataset, as described in Section 4.3.2. Transitions between pairs of 32 poses make up the majority of the dataset, providing only a small variation in hand poses. Although the dataset aims to cover the articulation space fully, it does not cover the variation of possible hand poses, which manifests poor performance on real data. Future work could

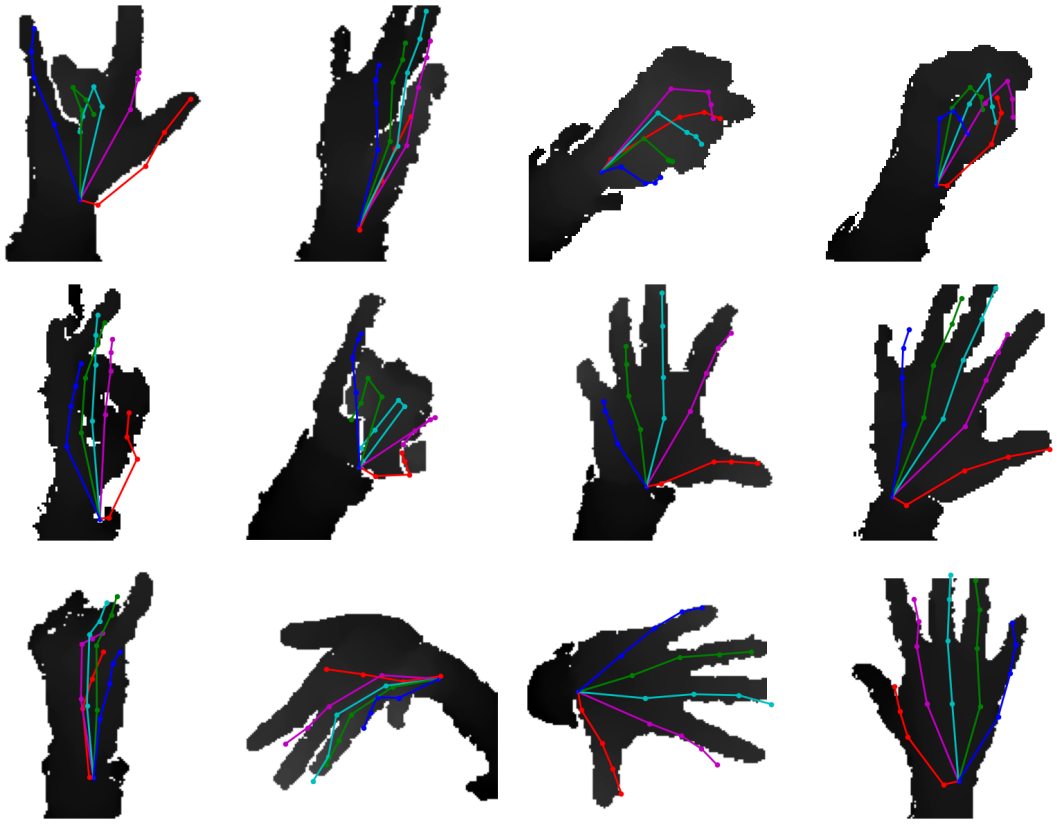


Figure 7.6: The estimator trained on the BigHand was tested real-time on images captured live. It seems the model poorly generalizes to poses on which it did not train. The bounding cube's side length was set to 160 mm.

explore other datasets, especially synthetic ones, or create new datasets that will contain wider variability of hand poses.

7.4 Gesture recognition on MSRA15 gesture dataset

Section 5.2.4 proposed a method for gesture recognition of static gestures. The method compares two hands by finding the joint relation errors between them. Summing these values results in an overall error score saying how much the poses differ—the higher the score, the more significant the difference between them. The gesture classification can be done by comparing the input image with all samples from the training dataset. The input image is classified as a gesture with the lowest error score.

The validity of this approach was verified on the MSRA15 gesture dataset, as shown in Figure 7.7. The training dataset was sampled randomly with the same number of samples for each gesture. The number of images in the training dataset was kept small to keep the computation fast and show that a small number of samples is enough for accurate classification. The rest of the images were used as a test dataset. The results demonstrate that the method is capable of precise classification given correct 3D skeletal information.

The need for finger length normalization was also verified, as illustrated in the figure. Without the normalization, the number of training samples needs to be higher to achieve

the same accuracy as with the normalization. Randomly sampling more images increases the probability of containing hands of the same size as in the test dataset, improving the overall accuracy. The mean finger length was determined on the MSRA dataset to be 65 mm.

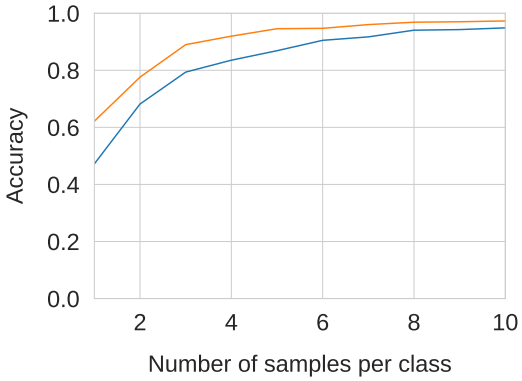


Figure 7.7: The proposed gesture recognition algorithm was evaluated on the ground truth labels of the MSRA15 dataset to classify 17 gestures. The figure demonstrates that a low number of training samples is sufficient to perform accurate classification. Applying finger length normalization produces higher accuracy (orange) than without this normalization (blue).

7.5 System evaluation

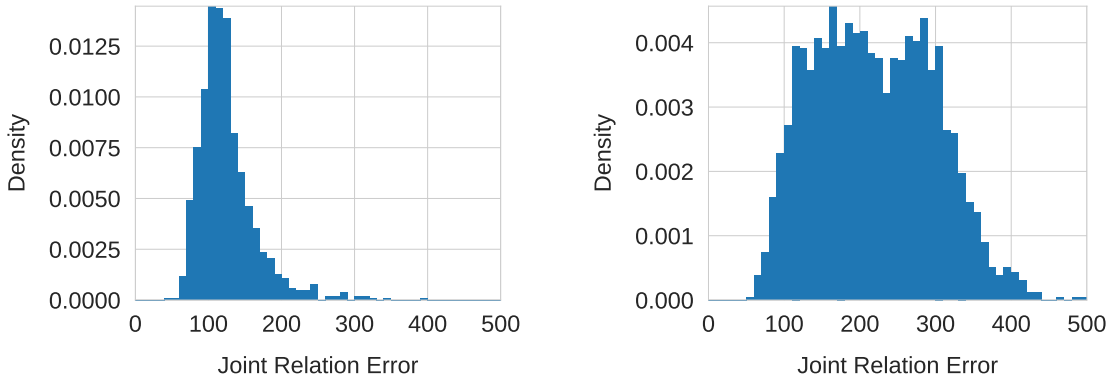
The system, whose implementation was described in Section 6.2, was evaluated on the custom dataset from Section 4.4 and tested in real-time.

7.5.1 Evaluation on a custom dataset

The system, proposed by Section 5.2, was evaluated on the custom dataset described in Section 4.4. It uses a hand estimator trained on the BigHand dataset with an increased number of features to 196 and the length of the bounding cube's side set to 160 mm. The preprocessing pipeline uses the center of mass to refine the detected bounding boxes.

In this system, the user specifies the target gesture. They scan the hand poses that they want to have as target gestures. Hence, the first step of the evaluation was the specification of the target gesture. The *gesture 1* (opened palm with fingers outstretched and apart), defined in Section 4.4, was chosen for the evaluation. A sequence consisting of 12 images was recorded for each hand. The hand was rotated in-plane during the recording to incorporate a wider variety of the same gesture. The system estimates the poses of the captured images and saves the locations of the 21 joints in world coordinates into files.

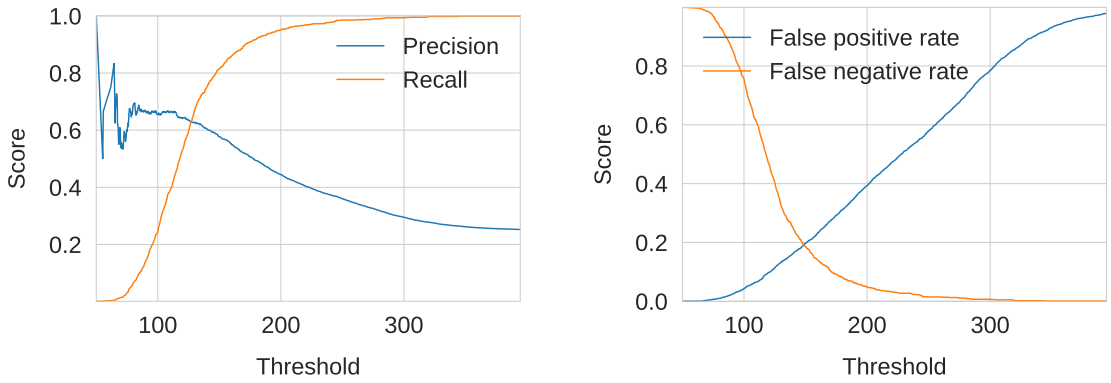
The second step of the evaluation is loading images of the custom dataset, estimating poses, and determining if the gesture is valid based on the joint relation error (JRE) metric and hand orientation difference. The JRE is proportional to the difference between poses, as Figure 7.8 demonstrates. The JRE for *gesture 1* from the custom dataset is distinctly lower in comparison to *non-gesture* poses. Ideally, the two density functions should not overlap to differentiate between *gesture 1* and *non-gesture* poses without error.



(a) (b)

Figure 7.8: The figures display density functions of joint relation error for hand poses from the custom dataset. The annotated images as *gesture 1* **(a)** show lower JRE than *non-gesture* poses **(b)**.

The choice of thresholds for the JRE and orientation difference reflects the model's performance. Figure 7.9 shows precision and recall metrics and false positives and negatives for different JRE thresholds. The precision does not exceed 0.7 while keeping the recall above 0.5. The high number of false positives limits the precision. The system should preferably eliminate false positives as much as possible despite increasing false negatives. A good choice of the threshold for JRE could be between 100 and 150. Nonetheless, the system's performance is low caused by the insufficient pose estimator's performance discussed in the previous section.

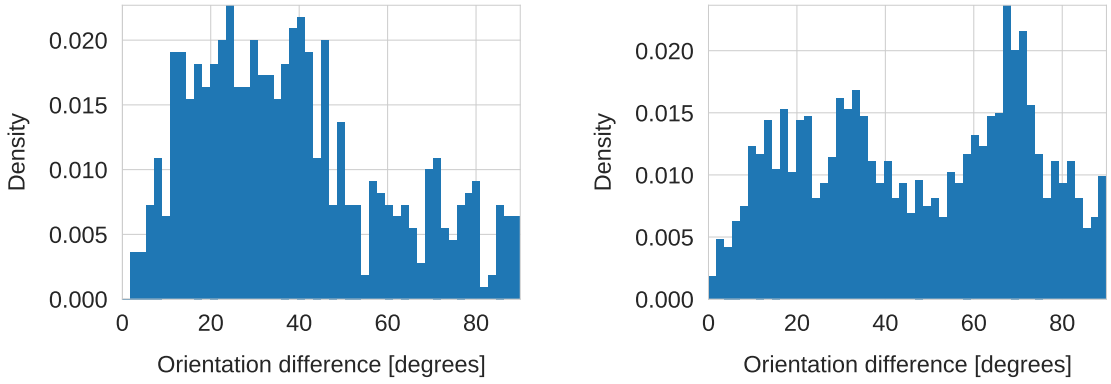


(a) Gesture 1 (b) Non-gesture

Figure 7.9: The figures display the results of the evaluation on the custom dataset. *gesture 1* was selected as the target gesture for recognition. The orientation difference was not taken into account. **(a)** shows precision and recall metrics for different thresholds, and **(b)** displays false positives and negatives.

The orientation difference appears to be substantially affected by poor skeletal estimation, as Figure 7.10 demonstrates. The orientation difference for images containing the

target gesture should not exceed 45 degrees, because the gesture was always captured at roughly the same orientation. The figure shows histograms only for poses of the right hand, for which the estimator produces less erroneous predictions than for the left hand.



(a) Gesture 1

(b) Non-gesture

Figure 7.10: The figures show density functions of orientation differences for poses in the custom dataset. Orientation differences for *gesture 1* **(a)** should not exceed 45 degrees, because the gesture was always captured at roughly the same orientation. Unfortunately, the estimator's poor performance highly affects the orientation, leading to differences above 45 degrees for the target gesture. Orientation differences for arbitrary poses **(b)** take up the whole orientation range because subjects randomly chose the poses' orientations.

On the whole, the system showed promising results on the custom dataset. Nonetheless, future work should concentrate on improving pose estimation as it is currently the system's limitation in terms of precision.

7.5.2 Testing real-time

Testing real-time uses a similar setup as the evaluation on the custom dataset in the previous section. The target gesture was selected to be *gesture 1* as before using the same database of the target gesture. The SR305 depth camera is used for capturing images, which are then compared to the gesture database and the results of gesture acceptance are plotted. The thresholds for JRE and orientation difference were selected to be 120 and 60, respectively.

The system works surprisingly well. A single misplaced finger causes the gesture to be categorized as a *non-gesture* pose, presented in Figure 7.11. Testing also revealed that the system's performance is far better for the right hand than for the left.

The real-time gesture recognition using Intel(R) Core(TM) i7-7500U 2.70GHz CPU achieves 4.75 FPS and 2.92 FPS if the result is plotted.

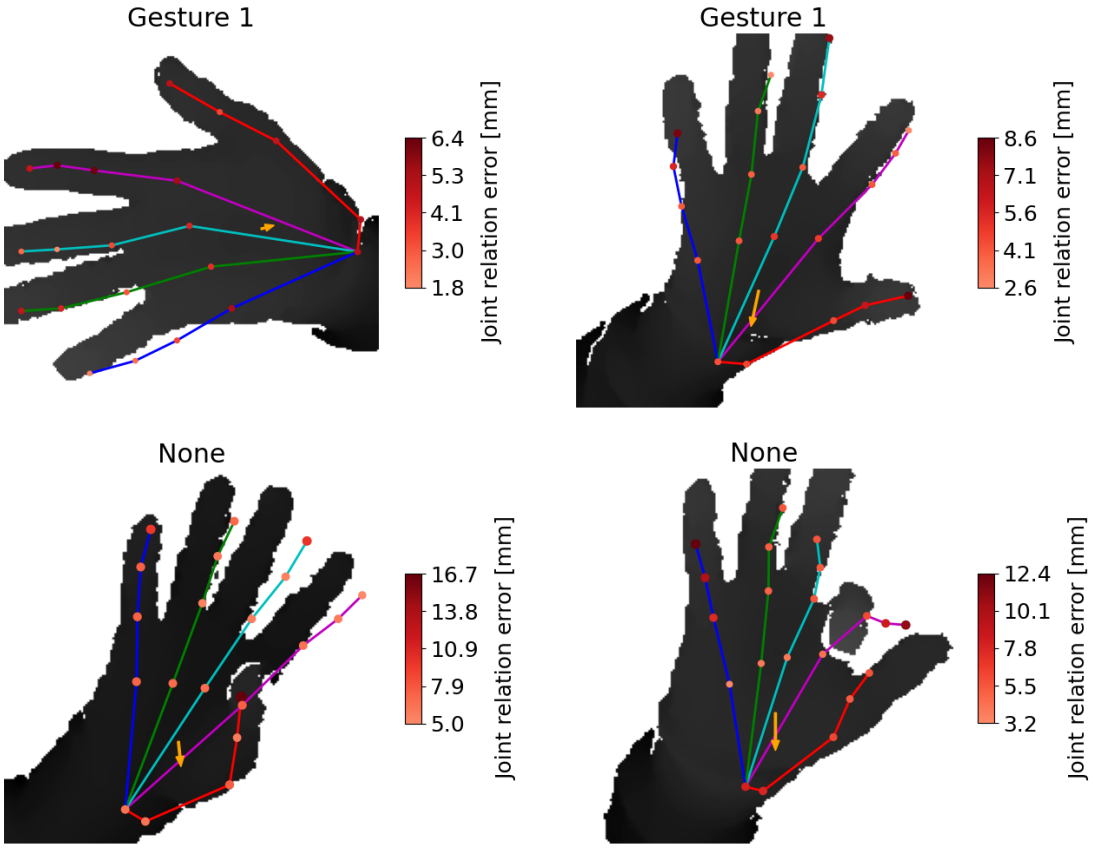


Figure 7.11: The figure depicts the system's output. Each joint is represented by a red dot whose size is proportional to the joint relation error. The yellow arrow indicates the palm's orientation. Each hand pose is assigned a gesture label. The top two images in the figure above were classified as *gesture 1*, while the bottom ones were classified as *non-gesture* poses.

Chapter 8

Conclusion

This work aimed to create a system for gesture recognition using skeletal information. The first part of the thesis presented state-of-the-art object detection and hand pose estimation methods along with a short review of gesture recognition. Available datasets for these tasks were listed as well. This work then proposed a system for gesture recognition from skeletal features represented by 21 key points. The developed system consists of three fundamental stages: detection, pose estimation, and gesture recognition. Each of these stages was evaluated separately on public datasets. A dataset consisting of four thousand images was were to evaluate the proposed system, which was also evaluated in real-time.

The Tiny YOLOv3 was selected for object detection for its speed and robustness. It was trained on the HandSeg dataset with great success, achieving an IoU score of 86.35 %. A series of preprocessing steps were developed for background removal and proper normalization of the image. It is the input of the hand pose estimator, JGR-P2O, which infers the precise position of the hand's skeleton. The evaluation demonstrated that the estimator's performance is highly dependent on the dataset. The low variety of poses in the dataset causes the model to generalize poorly to unseen poses in a natural environment. Several experiments were conducted, some of which improved the estimator's performance. On the MSRA dataset, the model reaches a mean joint error of 14.7 mm, while on the Bighand, a more complex dataset, achieves roughly 24.9 mm. Testing revealed that the estimator works better for the right hand than the left, which is due to the composition of the training dataset.

The proposed skeleton-based gesture recognition system allows the user to specify the target gesture and determine the hand's maximum rotation relative to that target gesture. On the other hand, the disadvantage of this solution is the need for accurate determination of the skeleton, which proves to be a challenging task.

Future work

Future work could try employing hand tracking for both object detection and hand pose estimation. The tracking might result in fewer false negatives during object detection and improve the inferred pose.

Furthermore, object detection could be achieved by extracting the closest contour, which many hand pose estimation techniques use. However, this technique assumes the hand is the closest object. The user would be free to choose which detection method better suits their use case.

The YOLOv3 predicts at multiple scales to detect objects of different sizes. An object's size in the image plane is dependant on the distance from the camera, which is unknown in color images, but it is known in depth images. Therefore, the architecture of YOLOv3 could be experimented with—instead of using bounding box priors of predefined sizes, it could adjust their size based on the depth. This idea is motivated by the fact that the hand has always relatively the same size and that it is the only object detected. The depth information could be utilized in the architecture of the object detector similarly to Deep Adaptive Neural Networks proposed by Kang et al. [29], which would remove the need for having more scale layers.

The HandSeg dataset contains annotations that differentiate between the left and right hand. The object detector could be adjusted to classify these two classes. The distinction would be helpful for the gesture recognition stage in telling whether the open palm faces the camera or if it is rotated away from the camera. The system cannot accurately perform this differentiation based on the skeletal information itself.

The most important limitation of the system is hand pose estimation because it poorly generalizes to unseen poses. Future work should concentrate mainly on this task. Section 7.3 suggested several ideas that might improve the model, such as removing the forearm during preprocessing, trying another pose estimation method, or training on another dataset. Other datasets could be explored, or a new one could be created specifically to cover an enormous number of hand poses and viewpoints. Using synthetic data is especially appealing for its potential to cover a large number of hand poses. I suggest creating a comparison of potential datasets similarly to Figure 4.3 by extracting the hand pose space and applying a dimensionality reduction technique.

Moreover, the estimated poses could be corrected using mathematical modeling of the hand's kinematics and dynamics. The system would refine the joints' locations by correcting the invalid locations that break the mathematical model.

Bibliography

[1] Arachchi, S. P. K., Hakim, N. L., Hsu, H.-H., Klimenko, S. V. and Shih, T. K. Real-Time Static and Dynamic Gesture Recognition Using Mixed Space Features for 3D Virtual World's Interactions. In: Barolli, L., Takizawa, M., Enokido, T., Ogiela, M. R., Ogiela, L. et al., ed. *32nd International Conference on Advanced Information Networking and Applications Workshops, AINA 2018 workshops, Krakow, Poland, May 16-18, 2018*. IEEE Computer Society, 2018, p. 627–632. DOI: 10.1109/WAINA.2018.00157. ISBN 978-1-5386-5396-8. Available at: https://doi.org/10.1109/WAINA.2018.00157.

[2] Armagan, A., Garcia-Hernando, G., Baek, S., Hampali, S., Rad, M. et al. Measuring Generalisation to Unseen Viewpoints, Articulations, Shapes and Objects for 3D Hand Pose Estimation under Hand-Object Interaction. *CoRR*. 2020, abs/2003.13764. Available at: https://arxiv.org/abs/2003.13764.

[3] Barsoum, E. Articulated Hand Pose Estimation Review. *CoRR*. 2016, abs/1604.06195. Available at: http://arxiv.org/abs/1604.06195.

[4] Bochkovskiy, A., Wang, C. and Liao, H. M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *CoRR*. 2020, abs/2004.10934, [cit. 2020-10-29]. Available at: https://arxiv.org/abs/2004.10934.

[5] Borji, A., Cheng, M., Jiang, H. and Li, J. Salient Object Detection: A Survey. *CoRR*. 2014, abs/1411.5878. Available at: http://arxiv.org/abs/1411.5878.

[6] campos), C. C. (https://math.stackexchange.com/users/517435/carlos. *Plane fitting using SVD normal vector* [Mathematics Stack Exchange]. [cit. 2021-04-02]. URL:https://math.stackexchange.com/q/2810220 (version: 2018-06-06). Available at: https://math.stackexchange.com/q/2810220.

[7] Chen, X., Wang, G., Guo, H. and Zhang, C. Pose guided structured region ensemble network for cascaded hand pose estimation. *Neurocomputing*. 2020, vol. 395, p. 138–149. DOI: 10.1016/j.neucom.2018.06.097. Available at: https://doi.org/10.1016/j.neucom.2018.06.097.

[8] Chen, X., Wang, G., Zhang, C., Kim, T. and Ji, X. SHPR-Net: Deep Semantic Hand Pose Regression From Point Clouds. *IEEE Access*. 2018, vol. 6, p. 43425–43439. DOI: 10.1109/ACCESS.2018.2863540. Available at: https://doi.org/10.1109/ACCESS.2018.2863540.

[9] Cheng, H., Jiang, X., Sun, Y. and Wang, J. Color image segmentation: Advances and prospects. *Pattern Recognition*. december 2001, vol. 34, p. 2259–2281. DOI:

10.1016/S0031-3203(00)00149-7. Available at: http://read.pudn.com/downloads152/doc/661708/01segmentadvances-pr.pdf.

[10] Czakon, J. *F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?* [online]. revised 2021-03-26 [cit. 2021-03-28]. Available at: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc.

[11] Du, K., Lin, X., Sun, Y. and Ma, X. CrossInfoNet: Multi-Task Information Sharing Based Hand Pose Estimation. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, p. 9896–9905. DOI: 10.1109/CVPR.2019.01013. ISBN 978-1-7281-3294-5. Available at: http://openaccess.thecvf.com/content_CVPR_2019/html/Du_CrossInfoNet_Multi-Task_Information_Sharing_Based_Hand_Pose_Estimation_CVPR_2019_paper.html.

[12] Düntsch, I. and Gediga, G. Confusion matrices and rough set data analysis. *CoRR*. 2019, abs/1902.01487. Available at: http://arxiv.org/abs/1902.01487.

[13] Fang, L., Liu, X., Liu, L., Xu, H. and Kang, W. JGR-P2O: Joint Graph Reasoning based Pixel-to-Offset Prediction Network for 3D Hand Pose Estimation from a Single Depth Image. *CoRR*. 2020, abs/2007.04646. Available at: https://arxiv.org/abs/2007.04646.

[14] Garcia-Hernando, G., Yuan, S., Baek, S. and Kim, T. First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations. *CoRR*. 2017, abs/1704.02463. Available at: http://arxiv.org/abs/1704.02463.

[15] Ge, L., Cai, Y., Weng, J. and Yuan, J. Hand PointNet: 3D Hand Pose Estimation Using Point Sets. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, p. 8417–8426. DOI: 10.1109/CVPR.2018.00878. ISBN 78-1-5386-6420-9. Available at: http://openaccess.thecvf.com/content_cvpr_2018/html/Ge_Hand_PointNet_3D_CVPR_2018_paper.html.

[16] Ge, L., Liang, H., Yuan, J. and Thalmann, D. 3D Convolutional Neural Networks for Efficient and Robust Hand Pose Estimation from Single Depth Images. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, p. 5679–5688. DOI: 10.1109/CVPR.2017.602. ISBN 978-1-5386-0457-1. Available at: https://doi.org/10.1109/CVPR.2017.602.

[17] Ge, L., Ren, Z. and Yuan, J. Point-to-Point Regression PointNet for 3D Hand Pose Estimation. In: Ferrari, V., Hebert, M., Sminchisescu, C. and Weiss, Y., ed. *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*. Springer, 2018, vol. 11217, p. 489–505. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-01261-8_29. Available at: https://doi.org/10.1007/978-3-030-01261-8_29.

[18] Girshick, R. B. Fast R-CNN. *CoRR* [[online]]. 2015, abs/1504.08083, 27. Sep 2015, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1504.08083.

[19] Girshick, R. B., Donahue, J., Darrell, T. and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR* [[online]]. 2013, abs/1311.2524, 22 Oct 2014, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1311.2524.

[20] Guo, H., Wang, G., Chen, X. and Zhang, C. Towards Good Practices for Deep 3D Hand Pose Estimation. *CoRR*. 2017, abs/1707.07248. Available at: http://arxiv.org/abs/1707.07248.

[21] Géron, A. *Hands-On Machine Learning with Scikit-Learn & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st ed. O'Reilly Media, March 2017. ISBN 978-1-491-96229-9.

[22] Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P. et al. Array programming with NumPy. *Nature*. Springer Science and Business Media LLC. Sep 2020, vol. 585, no. 7825, p. 357–362. DOI: 10.1038/s41586-020-2649-2. ISSN 1476-4687. Available at: http://dx.doi.org/10.1038/s41586-020-2649-2.

[23] Hartley, R. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. ISBN 978-0-521-54051-3.

[24] He, K., Gkioxari, G., Dollár, P. and Girshick, R. B. Mask R-CNN. *CoRR* [[online]]. 2017, abs/1703.06870, 24 Jan 2018, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1703.06870.

[25] He, K., Zhang, X., Ren, S. and Sun, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *CoRR* [[online]]. 2014, abs/1406.4729, 23 Apr 2015, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1406.4729.

[26] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. *CoRR*. 2015, abs/1512.03385. Available at: http://arxiv.org/abs/1512.03385.

[27] Huber, P. J. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*. Institute of Mathematical Statistics. 1964, vol. 35, no. 1, p. 73 – 101. DOI: 10.1214/aoms/1177703732. Available at: https://doi.org/10.1214/aoms/1177703732.

[28] III, J. S. S., Rogez, G., Yang, Y., Shotton, J. and Ramanan, D. Depth-based hand pose estimation: methods, data, and challenges. *CoRR*. 2015, abs/1504.06378. Available at: http://arxiv.org/abs/1504.06378.

[29] Kang, B., Lee, Y. and Nguyen, T. Q. Depth Adaptive Deep Neural Network for Semantic Segmentation. *CoRR*. 2017, abs/1708.01818. Available at: http://arxiv.org/abs/1708.01818.

[30] Li, B., Li, G., Sun, Y., Jiang, G., Kong, J. et al. A Review of Gesture Recognition Based on Computer Vision. In: Huang, Y., Wu, H., Liu, H. and Yin, Z., ed. *Intelligent Robotics and Applications - 10th International Conference, ICIRA 2017, Wuhan, China, August 16-18, 2017, Proceedings, Part I*. Springer, 2017, vol. 10462, p. 528–538. Lecture Notes in Computer Science. DOI:

10.1007/978-3-319-65289-4_50. Available at: https://doi.org/10.1007/978-3-319-65289-4_50.

[31] Li, S. and Lee, D. Point-To-Pose Voting Based Hand Pose Estimation Using Residual Permutation Equivariant Layer. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, p. 11927–11936. DOI: 10.1109/CVPR.2019.01220. ISBN 978-1-7281-3294-5. Available at: http://openaccess.thecvf.com/content_CVPR_2019/html/Li_Point-To-Pose_Voting_Based_Hand_Pose_Estimation_Using_Residual_Permutation_Equivariant_CVPR_2019_paper.html.

[32] Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B. et al. Feature Pyramid Networks for Object Detection. *CoRR* [[online]]. 2016, abs/1612.03144, 19 Apr 2017, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1612.03144.

[33] Lin, T., Goyal, P., Girshick, R. B., He, K. and Dollár, P. Focal Loss for Dense Object Detection. *CoRR* [[online]]. 2017, abs/1708.02002, 7 Feb 2018, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1708.02002.

[34] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E. et al. SSD: Single Shot MultiBox Detector. *CoRR* [[online]]. 2015, abs/1512.02325, 29 Dec 2016, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1512.02325.

[35] Maaten, L. van der. Learning a Parametric Embedding by Preserving Local Structure. In: Dyk, D. A. V. and Welling, M., ed. *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*. JMLR.org, 2009, vol. 5, p. 384–391. JMLR Proceedings. Available at: http://proceedings.mlr.press/v5/maaten09a.html.

[36] Malik, J., Abdelaziz, I., Elhayek, A., Shimada, S., Ali, S. A. et al. HandVoxNet: Deep Voxel-Based Network for 3D Hand Shape and Pose Estimation from a Single Depth Map. *CoRR*. 2020, abs/2004.01588. Available at: https://arxiv.org/abs/2004.01588.

[37] Malik, J., Elhayek, A., Nunnari, F. and Stricker, D. Simple and effective deep hand shape and pose regression from a single depth image. *Comput. Graph.* 2019, vol. 85, p. 85–91. DOI: 10.1016/j.cag.2019.10.002. Available at: https://doi.org/10.1016/j.cag.2019.10.002.

[38] Malik, J., Elhayek, A., Nunnari, F., Varanasi, K., Tamaddon, K. et al. DeepHPS: End-to-end Estimation of 3D Hand Pose and Shape by Learning from Synthetic Depth. *CoRR*. 2018, abs/1808.09208. Available at: http://arxiv.org/abs/1808.09208.

[39] Malireddi, S. R., Mueller, F., Oberweger, M., Bojja, A. K., Lepetit, V. et al. HandSeg: A Dataset for Hand Segmentation from Depth Images. *CoRR*. 2017, abs/1711.05944. Available at: http://arxiv.org/abs/1711.05944.

[40] Moon, G., Chang, J. Y. and Lee, K. M. V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation From a Single Depth

Map. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, p. 5079–5088. DOI: 10.1109/CVPR.2018.00533. ISBN 978-1-5386-6421-6. Available at: http://openaccess.thecvf.com/content_cvpr_2018/html/Moon_V2V-PoseNet_Voxel-to-Voxel_Prediction_CVPR_2018_paper.html.

[41] Newell, A., Yang, K. and Deng, J. Stacked Hourglass Networks for Human Pose Estimation. *CoRR*. 2016, abs/1603.06937. Available at: http://arxiv.org/abs/1603.06937.

[42] Oberweger, M. and Lepetit, V. DeepPrior++: Improving Fast and Accurate 3D Hand Pose Estimation. In: *2017 IEEE International Conference on Computer Vision Workshops, ICCV Workshops 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, p. 585–594. DOI: 10.1109/ICCVW.2017.75. ISBN 978-1-5386-1035-0. Available at: https://doi.org/10.1109/ICCVW.2017.75.

[43] Oberweger, M., Wohlhart, P. and Lepetit, V. Hands Deep in Deep Learning for Hand Pose Estimation. *CoRR*. 2015, abs/1502.06807. Available at: http://arxiv.org/abs/1502.06807.

[44] Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, vol. 9, no. 1, p. 62–66. DOI: 10.1109/TSMC.1979.4310076. ISSN 2168-2909.

[45] Qian, C., Sun, X., Wei, Y., Tang, X. and Sun, J. Realtime and Robust Hand Tracking from Depth. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2014, p. 1106–1113. DOI: 10.1109/CVPR.2014.145. ISBN 978-1-4799-5118-5. Available at: https://doi.org/10.1109/CVPR.2014.145.

[46] Redmon, J., Divvala, S. K., Girshick, R. B. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *CoRR* [[online]]. 2015, abs/1506.02640, 9 May 2016, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1506.02640.

[47] Redmon, J. and Farhadi, A. YOLOv3: An Incremental Improvement. *CoRR* [[online]]. 2018, abs/1804.02767, 8 Apr 2018, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1804.02767.

[48] Ren, S., He, K., Girshick, R. B. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR* [[online]]. 2015, abs/1506.01497, 6 Jan 2016, [cit. 2020-10-29]. Available at: http://arxiv.org/abs/1506.01497.

[49] Sharp, T., Keskin, C., Robertson, D. P., Taylor, J., Shotton, J. et al. Accurate, Robust, and Flexible Real-time Hand Tracking. In: Begole, B., Kim, J., Inkpen, K. and Woo, W., ed. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. ACM, 2015, p. 3633–3642. DOI: 10.1145/2702123.2702179. Available at: https://doi.org/10.1145/2702123.2702179.

[50] Sohn, M.-K., Lee, S.-H., Hwang, B., Kim, H. and Choi, H. Real-Time Hand Detection From a Single Depth Image by Per-Pixel Classification. *Journal of*

Industrial Information Technology and Application [[online]]. September 2017, Vol. 1, No. 2, [cit. 2020-07-14]. DOI: 10.22664/ISITA.2017.1.2.19. ISSN 2586-0852. Available at: http://jiita.org/vol1no201/.

[51] Subramanyam, V. S. *IOU (Intersection over Union)* [online]. revised 2021-01-17 [cit. 2021-04-23]. Available at: https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef.

[52] Sun, X., Wei, Y., Liang, S., Tang, X. and Sun, J. Cascaded hand pose regression. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, p. 824–832. DOI: 10.1109/CVPR.2015.7298683. ISBN 978-1-4673-6964-0. Available at: https://doi.org/10.1109/CVPR.2015.7298683.

[53] Tan, M., Pang, R. and Le, Q. V. EfficientDet: Scalable and Efficient Object Detection. *CoRR*. 2019, abs/1911.09070. Available at: http://arxiv.org/abs/1911.09070.

[54] Tang, D., Chang, H. J., Tejani, A. and Kim, T. Latent Regression Forest: Structured Estimation of 3D Articulated Hand Posture. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2014, p. 3786–3793. DOI: 10.1109/CVPR.2014.490. ISBN 978-1-4799-5118-5. Available at: https://doi.org/10.1109/CVPR.2014.490.

[55] Tompson, J., Stein, M., Lecun, Y. and Perlin, K.

[56] Tompson, J., Stein, M., Lecun, Y. and Perlin, K. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics*. Association for Computing Machinery (ACM). august 2014, vol. 33, no. 5. DOI: 10.1145/2629500. ISSN 0730-0301.

[57] Tzionas, D., Ballan, L., Srikantha, A., Aponte, P., Pollefeys, M. et al. Capturing Hands in Action using Discriminative Salient Points and Physics Simulation. *CoRR*. 2015, abs/1506.02178. Available at: http://arxiv.org/abs/1506.02178.

[58] Wan, C., Probst, T., Gool, L. V. and Yao, A. Dense 3D Regression for Hand Pose Estimation. *CoRR*. 2017, abs/1711.08996. Available at: http://arxiv.org/abs/1711.08996.

[59] Wetzler, A., Slossberg, R. and Kimmel, R. Rule Of Thumb: Deep derotation for improved fingertip detection. *CoRR*. 2015, abs/1507.05726. Available at: http://arxiv.org/abs/1507.05726.

[60] Wu, Y., Lim, J. and Yang, M. Online Object Tracking: A Benchmark. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*. IEEE Computer Society, 2013, p. 2411–2418. DOI: 10.1109/CVPR.2013.312. Available at: https://doi.org/10.1109/CVPR.2013.312.

[61] xinghaochen. *Awesome Hand Pose Estimation* [online]. GitHub, 2020 [cit. 2020-12-31]. Available at: https://github.com/xinghaochen/awesome-hand-pose-estimation.

[62] Xiong, F., Zhang, B., Xiao, Y., Cao, Z., Yu, T. et al. A2J: Anchor-to-Joint Regression Network for 3D Articulated Pose Estimation from a Single Depth Image. *CoRR*. 2019, abs/1908.09999. Available at: http://arxiv.org/abs/1908.09999.

[63] Xu, C. and Cheng, L. Efficient Hand Pose Estimation from a Single Depth Image. In: *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*. IEEE Computer Society, 2013, p. 3456–3462. DOI: 10.1109/ICCV.2013.429. ISBN 978-1-4799-2840-8. Available at: https://doi.org/10.1109/ICCV.2013.429.

[64] Ye, Q., Yuan, S. and Kim, T. Spatial Attention Deep Net with Partial PSO for Hierarchical Hybrid Hand Pose Estimation. *CoRR*. 2016, abs/1604.03334. Available at: http://arxiv.org/abs/1604.03334.

[65] Yuan, S., Garcia-Hernando, G., Stenger, B., Moon, G., Chang, J. Y. et al. 3D Hand Pose Estimation: From Current Achievements to Future Goals. *CoRR*. 2017, abs/1712.03917. Available at: http://arxiv.org/abs/1712.03917.

[66] Yuan, S., Ye, Q., Stenger, B., Jain, S. and Kim, T. BigHand2.2M Benchmark: Hand Pose Dataset and State of the Art Analysis. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, p. 2605–2613. DOI: 10.1109/CVPR.2017.279. ISBN 978-1-5386-0457-1. Available at: https://doi.org/10.1109/CVPR.2017.279.

[67] YunYang1994. *Tensorflow-yolov3* [online]. GitHub, 2020 [cit. 2020-8-10]. Available at: https://github.com/YunYang1994/tensorflow-yolov3.

[68] yy. *Object Detction #1: NMS* [online], 2. July 2018 [cit. 2021-03-17]. Available at: https://medium.com/@yusuken/object-detction-1-nms-ed00d16fdcf9.

[69] Zhang, T., Lin, H., Ju, Z. and Yang, C. Hand Gesture Recognition in Complex Background Based on Convolutional Pose Machine and Fuzzy Gaussian Mixture Models. *Int. J. Fuzzy Syst.* 2020, vol. 22, no. 4, p. 1330–1341. DOI: 10.1007/s40815-020-00825-w. Available at: https://doi.org/10.1007/s40815-020-00825-w.

[70] Zhao, Z., Zheng, P., Xu, S. and Wu, X. Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*. 2019, vol. 30, no. 11, p. 3212–3232. DOI: 10.1109/TNNLS.2018.2876865.

[71] Zhao, Z. yuan, Gao, W. lin, Zhu, M. miao and Yu, L. A Vision Based Method to Distinguish and Recognize Static and Dynamic Gesture. *Procedia Engineering*. 2012, vol. 29, p. 3065–3069. DOI: https://doi.org/10.1016/j.proeng.2012.01.441. ISSN 1877-7058. 2012 International Workshop on Information and Electronics Engineering. Available at: https://www.sciencedirect.com/science/article/pii/S1877705812004511.

[72] Zhou, X., Wan, Q., Zhang, W., Xue, X. and Wei, Y. Model-based Deep Hand Pose Estimation. *CoRR*. 2016, abs/1606.06854. Available at: http://arxiv.org/abs/1606.06854.

[73] Zimmermann, C. and Brox, T. Learning to Estimate 3D Hand Pose from Single RGB Images. *CoRR*. 2017, abs/1705.01389. Available at: http://arxiv.org/abs/1705.01389.

[74] Zou, Z., Shi, Z., Guo, Y. and Ye, J. Object Detection in 20 Years: A Survey. *CoRR*. 2019, abs/1905.05055. Available at: http://arxiv.org/abs/1905.05055.