Data simulation in deep learning-based human recognition

Data simulation in deep learning-based human recognition

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

M.Sc. Dennis Burgermeister (geb. Ludl)

aus Göppingen

Tübingen

2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	4. April 2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr.-Ing. Cristóbal Curio
2. Berichterstatter:	Prof. Dr. rer. nat. Andreas Schilling

Abstract

Human recognition is an important part of perception systems, such as those used in autonomous vehicles or robots. These systems often use deep neural networks for this purpose, which rely on large amounts of data that ideally cover various situations, movements, visual appearances, and interactions. However, obtaining such data is typically complex and expensive. In addition to raw data, labels are required to create training data for supervised learning. Thus, manual annotation of bounding boxes, keypoints, orientations, or actions performed is frequently necessary. This work addresses whether the laborious acquisition and creation of data can be simplified through targeted simulation. If data are generated in a simulation, information such as positions, dimensions, orientations, surfaces, and occlusions are already known, and appropriate labels can be generated automatically. A key question is whether deep neural networks, trained with simulated data, can be applied to real data. This work explores the use of simulated training data using examples from the field of pedestrian detection for autonomous vehicles. On the one hand, it is shown how existing systems can be improved by targeted retraining with simulation data, for example to better recognize corner cases. On the other hand, the work focuses on the generation of data that hardly or not occur at all in real standard datasets. It will be demonstrated how training data can be generated by targeted acquisition and combination of motion data and 3D models, which contain finely graded action labels to recognize even complex pedestrian situations. Through the diverse annotation data that simulations provide, it becomes possible to train deep neural networks for a wide variety of tasks with one dataset. In this work, such simulated data is used to train a novel deep multitask network that brings together diverse, previously mostly independently considered but related, tasks such as 2D and 3D human pose recognition and body and orientation estimation.

Kurzfassung

Die Erkennung von Menschen ist ein wichtiger Bestandteil von Perzeptionssysteme, wie sie beispielsweise in autonomen Fahrzeugen oder Robotern zum Einsatz kommen. Hierbei werden oft tiefe neuronale Netzwerke eingesetzt, die auf große Mengen an Daten angewiesen sind, welche idealerweise verschiedene Situationen, Bewegungen, visuelle Erscheinungsformen und Interaktionen abdecken. Die Erstellung dieser Daten ist jedoch oft aufwendig und teuer. Neben den Rohdaten werden auch Labels für diese Daten benötigt. So müssen zum Beispiel Bounding Boxes, menschliche Skelettrepräsentationen, Orientierungsangaben oder Aktionen manuell annotiert werden. Diese Arbeit widmet sich der Frage, ob diese aufwendige Beschaffung und Erstellung von Daten durch gezielte Simulation vereinfacht werden kann. Wenn Daten in einer Simulation erzeugt werden, hat dies den Vorteil, dass Informationen wie Position, Ausmaße, Orientierung, Oberflächen und Verdeckungen bereits bekannt sind und automatisch entsprechende Labels generiert werden können. Eine Kernfrage dabei ist, wie tiefe neuronale Netze, die mit simulierten Daten trainiert wurden, auf Realdaten angewendet werden können. Diese Arbeit untersucht den Einsatz von simulierten Trainingsdaten anhand von Beispielen aus dem Bereich der Fußgängererkennung für autonome Fahrzeuge. Dabei wird zum einen betrachtet, wie existierende Systeme durch gezieltes Nachtrainieren mit Simulationsdaten verbessert werden können, um etwa Randfälle, sogenannte Corner Cases, besser zu erkennen. Zum anderen liegt ein Schwerpunkt der Arbeit auf der Generierung von Daten, die in realen Standarddatensätzen kaum oder gar nicht vorkommen. Es wird aufgezeigt, wie durch gezielte Aufnahme und Kombination von Bewegungsdaten und 3D Modellen Trainingsdaten generiert werden können, die fein abgestufte Aktionslabels beinhalten, um auch komplexe Fußgängersituationen zu erkennen. Durch die vielfältigen Annotationsdaten, die Simulationen liefern, wird es möglich Netzwerke für verschiedenste Aufgaben mit einem Datensatz zu trainieren. In dieser Arbeit wird dies genutzt, um ein neuartiges, tiefes Multi-Task Netzwerk zu trainieren, das diverse, bisher meist unabhängig betrachtete, aber zusammenhängende, Aufgaben, wie die 2D und 3D Mensch-Posenerkennung sowie die Körper und die Orientierungsschätzung zusammenzubringen.

Acknowledgments

I would like to take this opportunity to thank several people who have supported me during my time as a PhD student. Foremost, I would like to thank my colleagues, and especially Thomas Gulde, who always supported me practically, were available for technical discussion and, finally, the numerous joint activities in our spare time. Furthermore, I would like to thank Andreas Schilling for organizing and conducting various doctoral seminars in which I was able to participate as an external doctoral student and which always encouraged professional exchange across project boundaries. In addition to the professional support, I would also like to thank my family, who have been by my side and supported me throughout this time. I would also like to thank the Cognitive Systems Research Group at Reutlingen University, where I was able to realize this work and which provided the necessary infrastructure for the implementation of various research ideas. A special thanks go to the leader of the Cognitive Systems research group and my PhD supervisor Cristóbal Curio. Without his support this work would not have been possible. Thank you for the discussions, suggestions, conference visits as well as the possibility to always broaden my horizon.

Parts of this work and especially the experiments in sections 4.1, 4.2 and 4.3 was sponsored by BMBF project *OFP* (16EMO0114) and *MoCap 4.0* (03FH015IN6). Further parts, especially on the Motion Capture Manager's development and the PedRecNet, were funded in a research cooperation by the Continental AG. Parts of the GPU hardware used in this work are sponsored through NVIDIA's academic program.

Contents

List of Figures **xv**

List of Tables **xix**

Glossary **xxi**

Acronyms **xxiii**

1 Introduction **1**

1.1 Contribution 3
1.2 Chapter Overview 4
1.3 Related Work 4
1.3.1 Classification 4
1.3.2 Detection 5
1.3.3 2D Human Pose Estimation 6
1.3.4 3D Human Pose Estimation 8
1.3.5 Body and Head Orientation Estimation 9
1.3.6 Action Recognition 11
1.3.7 Simulation in Deep Learning 12

2 Deep Learning **15**

2.1 Artificial Neural Networks 16
2.1.1 Activation Functions 18
2.1.2 Fully Connected Layer 19
2.2 Training 19
2.2.1 Weight Initialization 20
2.2.2 Loss Functions 21
2.2.3 Optimization 23
2.2.4 Batch Normalization 24
2.2.5 Regularization 25
2.2.6 Convolutional Neural Networks 27
2.3 Metrics 31
2.3.1 Classification 31
2.3.2 Detection 35

2.3.3 2D Human Pose Estimation 37
2.3.4 3D Human Pose Estimation 39

3 Simulation 41
3.1 General Requirements 41
3.1.1 Required Data in the Simulation Framework 42
3.1.2 Data Generation 42
3.1.3 Ground Truth Labels 42
3.2 Simulation Framework 44
3.3 Required Data To Generate Simulations 45
3.4 3D Models 46
3.4.1 3D Environments 46
3.4.2 3D Objects 46
3.4.3 3D Humans 47
3.5 Animations 48
3.5.1 Motion Capture Manager 52
3.6 Asset Database 57
3.7 Simulation 59
3.7.1 Asset Import 59
3.7.2 Content Generation 61
3.7.3 Virtual Sensors 63
3.7.4 Ground Truth Data Generation 65

4 Applications 71
4.1 Improving the Detection of Corner Cases in 2D Human Pose Estimation 73
4.1.1 Data Generation 74
4.1.2 Experiments 75
4.1.3 Discussion 82
4.2 Pose-based Action Recognition 83
4.2.1 Recognition Pipeline 85
4.2.2 Pose-based Action Recognition 88
4.2.3 Datasets 93
4.2.4 Training 94
4.2.5 Experiments 96
4.2.6 Discussion 98
4.3 Action Recognition Using Only Simulated Training Data 99
4.3.1 Simulated Data 99
4.3.2 Virtual Content Generation 99
4.3.3 Datasets 102
4.3.4 Pose Recognition Algorithm 104
4.3.5 Action Recognition Algorithm 104

4.3.6 Data Preprocessing 105
4.3.7 Results 105
4.3.8 Discussion 110
4.4 PedRec: Multi-Task Pedestrian Recognition 112
4.4.1 Definitions 112
4.4.2 Network Architecture 114
4.4.3 Heatmap-based Workflow 116
4.4.4 Datasets 122
4.4.5 Training Procedure 127
4.4.6 Results 129
4.4.7 Discussion 141
4.5 EHPI3D action recognition framework based on 3D human pose . . 143
4.5.1 Generation of EHPI3Ds 143
4.5.2 Unit Skeleton 144
4.5.3 Training 146
4.5.4 Experiments 146
4.5.5 Discussion 151
4.6 Conclusions from all experiments 151

5 Conclusion **153**

A Tables **157**
A.1 Reutlingen University - Motion Capture Lab: Hardware Overview . 157
A.2 SIM-C01: Record Catalogue 157
A.3 SIM-C01: Used Accessories 159
A.4 EHPI3D: Unit Skeleton 159

B Listings **161**
B.1 Character Description File 161
B.2 Motion Capture Description File 161

C Figures **165**
C.1 EHPI3D Examples 165

Bibliography **169**

List of Figures

1.1 Example Annotations for Human Behavior 2

2.1 Classic Machine Learning Pipeline 16
2.2 Deep Learning Pipeline 17
2.3 Artificial Neuron 17
2.4 Gradient Descent and the Influence of the Momentum 20
2.5 Visualization of the Influence of Weight Decay as a Regularization Term 26
2.6 Overview of Common Convolutional Layer Parameters 27
2.7 convolutional neural network (CNN) Kernels on RGB Inputs 28
2.8 Transposed Convolution 2D 30
2.9 Transposed Convolution 2D - Convolution 31
2.10 Three Class Confusion Matrix 33
2.11 Bounding Box intersection over union (IoU) 35
2.12 Example of a Precision-Recall Curve 36
2.13 2D Human Pose Estimation Example 38

3.1 Simulation Framework Overview 45
3.2 Example 3D Models From Different Sources 47
3.3 Motion Capture Session Procedure 51
3.4 Reutlingen University Motion Capture Lab: Participants' View . . 53
3.5 Motion Capture Manager: Sequence Conditions During a Record . 56
3.6 Action Labeling Tool 57
3.7 3D Human Avatar and Motion Filter Options 60
3.8 Retargeting Issue 61
3.9 Multiple Cars in a Scene 63
3.10 Camera Sphere 64
3.11 Example of a simulated scene with visualized ground truth. 66
3.12 Skeleton Structures 67
3.13 PedRec Skeleton Structure 68

4.1 Corner Case: Handstand 73
4.2 Corner Cases: Training Data Example 75
4.3 OpenPose Network Outputs 77

4.4 Corner Case: Before and After 78
4.5 Corner Cases: Detection Issues 79
4.6 Corner Cases: PCK Results on COCO2017 80
4.7 Corner Cases: Before and After Results in Different Situations . . . 81
4.8 Encoded Human Pose Image (EHPI) Generation 84
4.9 EHPI Action Recognition Pipeline 85
4.10 EHPI: CNN Architecture 89
4.11 EHPI Examples from Three Different Actions 91
4.12 EHPI: Example Waving Sequence 92
4.13 EHPI: Example of Simulated Data from Different Perspectives . . . 95
4.14 ActionSim: Simulation Environment 100
4.15 ActionSim: Generated Ground Truth Data 101
4.16 ActionSim: Virtual Camera Setup 103
4.17 ActionSim: Action Confusion Matrix 105
4.18 Idle Action Misclassified as Walking 107
4.19 Example of Confusion Between Idling and Sitting Actions 108
4.20 Simulated Waving Out Scene 110
4.21 Local Skeleton 3D Coordinate System 113
4.22 Visualization of the orientation estimation for each, the body and the head. The orange dot shows an example point on a 3D sphere, visualizing an orientation. The standard notation from ISO 80000-2:2019 [135] is used for spherical coordinates. As only the polar angle θ and azimuthal angle φ are required, a unit sphere and with $r = 1$ was used. 113
4.23 Simplified PedRecNet Architecture 114
4.24 Heatmap-based Prediction Pipeline: Encoder - Decoder 116
4.25 Softmax Visualization 118
4.26 Comparison of Softargmax Outputs 118
4.27 Human Pose: Softmax Heatmaps 119
4.28 Human Body and Head Orientation: Softmax Heatmaps 120
4.29 Human Pose: Softmax Heatmaps with Peakiness Constraint 120
4.30 Polar Coordinate Represented as Cartesian Coordinates 122
4.31 Distribution of Body θ Orientations 123
4.32 Motion Capture Manager: Record Duration Distribution 126
4.33 2D Human Pose Estimation: Real-world example 133
4.34 3D Joint Distances on the H36M [48] Validation Dataset 134
4.35 3D Joint Distances on the SIM-C01 Validation Dataset 135
4.36 3D Human Pose Estimation ‘in the wild’: PedRecNet Examples . . 136
4.37 3D Human Pose Estimation ‘in the wild’: PedRecNet False Predictions 137
4.38 MEBOW [65] Validation Dataset: Wrong Labels 139
4.39 MEBOW [65] Validation Dataset: Occluded or Hard to See People 139
4.40 MEBOW [65] Validation Dataset: Wrong Orientation Predictions . 140

4.41 Head and Body Orientation Estimation ‘in the wild’: Examples . . 142
4.42 Example of the EHPI3D Structure 144
4.43 Human Body Proportions 145
4.44 Human Head Proportions 145
4.45 EHPI3D Examples 150

5.1 PedRec application 154

C.1 EHPI3D Examples (A) 166
C.2 EHPI3D Examples (B) 167

List of Tables

2.1 Example Activation Functions 18
2.2 Binary Confusion Matrix 32

3.1 Asset Database: Data, Metadata, and Sources 58

4.1 Corner Cases: RARESIM Results 76
4.2 Corner Cases: COCO 2017 Results 80
4.3 EHPI: JHMDB Results 97
4.4 EHPI: Parking Lot Use Case Results 97
4.5 ActionSim Results on Lab Data 106
4.6 ActionSim Results on Unrelated Real Data 109
4.7 Dataset Label Support 122
4.8 SIM-C01 - Number of Samples Per Action 124
4.9 Used 3D Models in the Simulation Datasets 126
4.10 Dataset - Statistics 127
4.11 Comparison of Used Training Datasets on COCO [25] val2017 Results 130
4.12 2D Human Pose Estimation Results: COCO [25] Skeleton 131
4.13 2D Human Pose Estimation Results: H36M [48] Skeleton 131
4.14 2D Human Pose Estimation Results: H36M [48] Extended Skeleton 132
4.15 2D Human Pose Estimation Results: PedRec Skeleton 132
4.16 H36M [48] Results 133
4.17 Human Body Orientation (φ) Results 138
4.18 PedRecNet: Head Orientation Test Results 140
4.19 EHPI3D Action Recognition Results on the SIM-C01V Dataset . . 147
4.20 EHPI3D Action Recognition Results per Action 149

Glossary

artificial neural network Biologically inspired neural networks which are able to learn tasks by adapting their internal parameters based on external signals. plural. xxi, xxiii

deep neural network A category of describing artificial neural networks with many layers, which do not directly try to replicate biolocial neural networks [1, p. 13]. The number of layers required to call a network deep is not defined, in this work a deep neural network consists of more than two layers. xxiii

LiDAR LiDAR is a portmanteau of light and radar. It describes devices which send out laser beams and measure the reflected light to get the return time as well as the wavelength. Using a rasterization of this lazerbeams enables the generation of 3D point clouds. . 41

one hot vector A binary vector with $n \in \mathbb{N}_{>0}$ fields representing categories. Only one of the values is set to one. Is used for example, in classification where the field corresponding to the correct class for a given input is set to one. 15

unique identifier A identifier which is unique among all identifier used for entities of a given set. Unique identifiers in this work consist of strings and / or numbers. plural. xxiv

Acronyms

ANN artificial neural network. 16, 20

AP average precision. 36, 39, 76, 130

BCE binary cross entropy loss. 22, 119–121, 146

CE cross entropy loss. 22

CNN convolutional neural network. xv, xvi, 11, 27–29

DNN deep neural network. 5, 15, 48, 71, 152, *Glossary:* deep neural network

DT decision tree. 15, 16

EHPI Encoded Human Pose Image. xvi, xix, 83, 84, 89–93, 96–98

FCNN fully connected neural network. 16, 29

FLIC Frames Labeled In Cinema. 7

GD gradient descent. 19, 20, 23

IMU inertial measurement unit. 9, 10, 49

IoU intersection over union. xv, 35, 36, 39

ITSC Intelligent Transportation Systems Conference. 3

LSP Leeds Sports Pose. 7

LSTM long-short term memory. 3, 19

mAP mean average precision. 35, 37, 76

mBA mean balanced accuracy. 146

MBGD	mini-batch gradient descent. 23, 24
MPJPE	mean per joint position error. 40, 133, 134
NMS	Non-maximum suppression. 115–117
OKS	object keypoint similarity. 38, 39, 43, 78, 130
PAF	part affinity field. 7
PCK	percentage of correct keypoints. 37–40, 78, 130–132
PDJ	percentage of detected joints. 38, 39
RBM	restricted boltzmann machine. 18
ReLU	rectified linear unit. 18, 21
ROI	region of interest. 5, 6
SGD	stochastic gradient descent. 23, 24
SOTA	state-of-the-art. 1, 3–5, 11, 18, 73, 74, 76, 82, 83, 86, 96, 98, 129, 141, 153
SVM	support vector machine. 15, 16
UID	unique identifier. 50, 52, 55, 56, 58, 63, 125, *Glossary:* unique identifier
YOLO	You Only Look Once. 6

Chapter 1

Introduction

Autonomous systems act in various areas, which contain miscellaneous direct and indirect interactions with humans. Examples range from autonomous vehicles over industrial and household robots to intelligent prostheses. There is a consensus that the machine understanding of humans and their behavior is one of the main challenges for autonomous systems [2]. Autonomous systems usually utilize sensor technology to perceive their environment. Based on the processed sensor information, those systems have to perform various tasks like detecting humans, handing a human worker a component, or prevent a collision with a human. Most state-of-the-art (SOTA) perceptual algorithms are data-driven and trained under supervision (see chapter 2). The performance of such data-driven algorithms highly depends on the quality of the annotated sensor data available at training time. The training data should contain a considerable variety of annotated data to enable the autonomous system to fulfill certain tasks. These annotations can range from individual bounding boxes over 2D and 3D human pose information to the intention of individuals. The collection and annotation of real sensor data are time-consuming. For example, suppose real sensor data is recorded from a car driving through various cities while recording pedestrians. In that case, it can be manually annotated where which human is at what time and what a human is doing. By carefully observing a longer time range, it may also be possible to answer why a person is doing something. For example, if someone is walking on the sidewalk and looks back because he wants to cross the street (see Figure 1.1).

Since the recording and annotation of such training data are correspondingly complex and thus expensive, standard datasets are usually used to train algorithms. However, these standard datasets do not contain data for all possible situations. Examples are rare poses such as handstands or certain actions such as waving a vehicle out. Another problem in standard datasets is that the more complex the annotation is, the fewer data is typically available (see section 1.3). Even if a dataset with appropriate annotations is available, they are typically hand-labeled and may contain errors. For example, in recent work, Northcutt *et al.* investigated the presence of errors in training [3] and test [4] dataset labels. They estimate

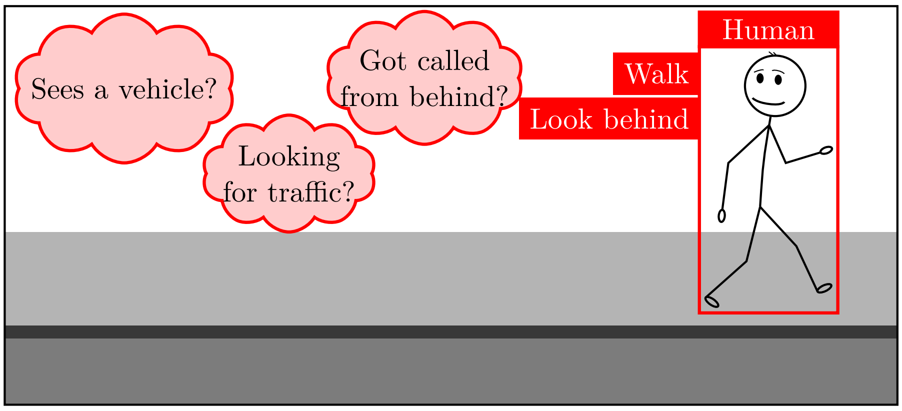


Figure 1.1: Exemplary annotations made on a single frame, observing a human on a sidewalk. Labeling of why the human looks around is not possible without further context.

an average of 3.4% errors in those labels, with up to 10% errors in one of the ten datasets examined [4, p. 6].

One possibility to quickly generate sensor data in different situations is simulation, which can also automatically provide complex annotation data. This work aims to create a simulation framework to show how simulated data can be generated with as little manual effort as possible, and how this simulation data can be used to train and support deep learning applications. These studies are to be conducted in the context of human detection, mainly in the context of autonomous driving. For this purpose, algorithms will be adapted and developed to capture various characteristics of people recorded with camera sensors. On the one hand, it will be considered how standard tasks, such as 2D human pose recognition, can be improved by generating data, that is not available in standard datasets, through a simulation. Furthermore, it will be considered how simulations can be applied in the context of special requirements in algorithm development. Action recognition serves as an example, which often requires special motion sequences not available in standard datasets. One significant issue in deep learning applications is the collection of data with complex labels, such as 3D human pose data or body and head orientation information. Those ground truths are hard to annotate manually in real data. Therefore, this thesis will investigate how simulated data with such complex labels can be used to train algorithms that can be applied to real data subsequently. Since human actions are highly dynamic, it is important to predict the actions correctly and in real-time. Thus, the algorithms used and developed in this work focus on runtime efficiency and accuracy.

1.1 Contribution

In [5] it is shown that a current SOTA pose recognition algorithm has problems recognizing corner case poses. An example of such a pose is the handstand. It was investigated to what extent simulated data can be used to adapt the algorithm to improve the recognition rate of these poses. A key point of this work is that data about a given condition may already exist in other domains and may be used to solve problems for which no data exists in the target domain. There are various motion capture databases in which poses, such as a handstand, are available in various executions that are often not or only strongly underrepresented in video and image datasets. By using these data in simulations, new sensor data can be generated quickly and without great manual effort, with which a corresponding algorithm can be trained. With this method, the recognition rate of human joints in rarely occurring poses could be increased by 13.7%. The work was awarded a **Runner-Up Best Paper Award** at the IEEE Intelligent Transportation Systems Conference (ITSC) 2018. The co-authors supported the author in managing and recording the data. The manuscript was written entirely by the author of this thesis. This work is shown in section 4.1.

A system for pose-based action recognition using the example of an autonomous valet parking function was presented in [6]. A deep learning-based architecture was designed to detect pedestrians in real-time, estimate their pose and determine an action based on the pose data. The system was used in an actual test vehicle to detect pedestrians and to detect if they wave a parking vehicle out of a parking lot. The architecture was also compared to other pose-based algorithms on a standard dataset and achieved a 3.5% higher accuracy in detecting actions. For the valet parking use case, it was also examined whether the action recognition can also be trained with simulation data. A recognition rate of 81.48% was achieved. This is clearly below the recognition rate of 99.26% when including real training data, but showed that a system could also be trained with simulation data. In this work, the co-authors supported the recording, administration, and processing of motion capture, 3D scan, and camera sensor data. The manuscript was written entirely by the author of this thesis. This work is shown in section 4.2

We further evaluated the approach of training pose-based action recognition algorithms with simulated data only in [7]. The main idea behind it was to recreate a laboratory scenario in a simulated environment, train a pose-based action recognition algorithm with the simulated data, and test the trained algorithm on real data. With the same camera positions and perspectives in the simulated environment, it is expected to reach a high recognition rate, assuming that the pose recognition algorithms perform similarly on simulated and real data. In the experiment, we recorded a participant in our motion capture laboratory who performed five different actions. A long-short term memory (LSTM) network was used to recognize actions based on pose-data. The LSTM network was able to recognize all sequences correctly. The recognition rate for each frame in the video was 98.52%, which shows that in the laboratory experiment, no domain shift issues occurred when working with pose-based action recognition algorithms. The co-authors supported the recording, administration, and processing of motion capture, 3D scan, and camera sensor data. The manuscript was written entirely by the author of this thesis. This work is shown in section 4.3.

Another paper introduced the PedRecNet multi-task network at IEEE Intelligent Vehicles 2022 [8]. The paper extends the 2D human pose estimation approach from [6] with 3D pose estimation and body and head orientation estimation. In doing so, the individual tasks exhibit recognition performance comparable to the current SOTA. The system supports the recognition of multiple persons simultaneously in real-time (>20 FPS). The architecture is deliberately kept simple, allowing the network to be easily adapted, modified and extended. The manuscript was written entirely by the author of this thesis. This work is described in detail in section 4.4.

1.2 Chapter Overview

The introduction and related work in chapter 1 is followed by required basics of deep learning applications in chapter 2. Chapter 3 provides information about the approach to simulated data and related work. Chapter 4 shows various applications of simulated data, generated with the developed simulation framework. The chapter shows the application of simulated training data in various deep learning applications and contains various experiments demonstrating the effect of such simulated training data. The work is concluded in chapter 5.

1.3 Related Work

The following literature review categorizes current deep learning-based computer vision algorithms applied in human recognition and used in this work. This section also includes information about the required annotated sensor data to train deep learning algorithms in the given category.

1.3.1 Classification

Classification algorithms indicate whether a certain object class can be found in sensor information. Classification is often one of the first steps in a detection pipeline to determine if a human is in an image or if a certain part of an image represents a human (see detection). A major classification milestone is AlexNet [9], which outperformed the SOTA in the ImageNet classification challenge (see datasets) in 2012 by over 10% accuracy through the use of a deep convolutional neural network (see section 2.2.6). The following approaches are based on the same foundation of deep neural networks containing convolutional layers. One frequently used network architecture is VGG [10], which has a reduced number of parameters compared to AlexNet due to fixed kernel sizes. ResNet [11] is another network architecture, which is also the basis of most networks in this work (see sections 4.2, 4.3 and 4.4). It enables deeper networks as it reduces the vanishing gradient problem (see section 2.2.1) by shortcut connections. Human classification performance is approximately at a top-5 accuracy of 5.1% [12, p. 31] and is even surpassed by current SOTA approaches which, for example, use neural architecture search to let a neural network search for optimal network architectures [13] or by replacing spatial convolutions with self-attention mechanisms [14]. Many other approaches are similar in structure and differ in details.

Datasets Various datasets contain classification labels, many of them are specialized on tasks like the classification of numbers[1], cars[2] or bees[3]. There are also various general purpose datasets containing many classes [15, 16, 17], but the most widely used dataset is ImageNet [16], which contains over $14,000,000$ images and $1,000$ classes[4]. Most deep neural networks (DNNs) in this work are pre-trained on ImageNet, due to the robust features obtained from the huge amount of data. Those pre-trained network parts are mostly used for feature extraction and fine-tuned to specific tasks, such as the 2D and 3D human pose estimation algorithms in this work.

Annotation Data For this work, the ResNet architecture is most relevant. It is mostly used as a feature extractor before further applications like 2D human pose estimation or 3D human pose estimation. The requirements for a simulation for classification tasks are simple. An image of a camera sensor must be output and provided with one or more labels that indicate which objects are in the image. Classification is often a subtask of other algorithms, for example, in detection tasks.

1.3.2 Detection

Detection algorithms provide the position and dimensions of objects in an image using a bounding box and classify the bounding box contents. The detection area can be roughly divided into two categories. The first category is based on the fact that different methods create so-called region of interest (ROI) proposals. This is a more extensive set of bounding boxes with different positions and dimensions in

[1]`http://yann.lecun.com/exdb/mnist/` (accessed on 2022-03-06)
[2]`https://ai.stanford.edu/~jkrause/cars/car_dataset.html` (accessed on 2022-03-06)
[3]`https://www.kaggle.com/jenny18/honey-bee-annotated-images` (accessed on 2022-03-06)
[4]`http://image-net.org/` (accessed on 2022-03-06)

the image, which tend to contain an object. On these ROIs, a classification is then executed. If the network returns a certain probability for containing an object with a class label, the bounding box proposal is accepted and output with the class label. This category includes, for example, R-CNN [18] which performs feature extraction on the bounding box for each ROI proposal and then classifies it. Its successors like Fast(er)-RCNN [19, 20] perform a feature extraction only once on the whole image, and the ROI proposals are generated on the feature vector. Since the feature extraction only takes place on the full image, this speeds up the process.

The second category consists of algorithms that do not require ROI proposals. They regress the position and extent of bounding boxes. These approaches include, for example, the You Only Look Once (YOLO) architecture [21]. Here, the image is divided into a grid of a certain size. Afterward, bounding boxes are regressed, and their contents are classified. The algorithm in section 4.2 uses YOLOv3 as a feature extractor before 2D human pose estimation, which is a further development of the original YOLO architecture that additionally includes skip connections, residual blocks, upsampling, and generally more layers. Furthermore, YOLOv3 detects on three different scaled feature maps to better detect small objects. Another YOLO version is YOLOv4 [22], which is used in section 4.4. In this version, adjustments have been made to the training process and minor architectural changes, but nothing has changed in the basic approach.

Datasets As for classification, there are many datasets for object detection specialized on certain tasks. Some popular datasets in the autonomous driving related field are the KITTI object detection dataset [23] and the Pascal VOC dataset [24]. One large and widely used dataset is the COCO dataset [25] which contains over $200,000$ labeled images with over $1,500,000$ object instances in 80 object categories[5]. The COCO dataset is used to pre-train the object detection algorithms used in sections 4.2 and 4.4.

Annotation Data Since further processing is based on bounding boxes, detection algorithms are always used as the first step of a detection pipeline in this work. As ground truth, detection algorithms require a list of contained objects and the bounding box of these objects for each camera image.

1.3.3 2D Human Pose Estimation

In two-dimensional pose recognition, the positions of human joints of a human skeleton are recognized and represented as image coordinates x and y. These are connected according to a defined skeleton configuration. The term *limb* is usually used for these connections, since these are mostly representations of human bones.

[5]`https://cocodataset.org/` (accessed on 2022-03-06)

An example would be the limb 'right femur', which connects the two joints 'right hip' and 'right knee'. One way to categorize 2D human pose algorithms is by dividing them into bottom-up or top-down approaches. Bottom-up approaches first detect all joints in an image and try to connect them to individual skeletons afterward. One possible bottom-up approach is first to detect body parts in the image and afterward assign the single body parts to skeletons using integer linear programming [26]. OpenPose [27] is another bottom-up approach, which was used in the experiments described in section 4.1. In this approach, the network predicts joint heatmaps over the full image and also outputs so-called part affinity fields (PAFs). Those PAFs are used to indicate the mapping between two joints. For example, it should output high peaks on the limb between the right knee and the right ankle of a skeleton, but only if both joints belong to the same human body. Top-down approaches first detect humans in an image and afterward recognize the joints of the detected humans one by one. Stacked hourglasses contain multiple down- and upsampling modules that capture information on multiple scales and can produce a heatmap for each joint. Using the max activation on this heatmap results in the joint coordinates. The activation strength can be used to estimate the presence or absence of a joint [28]. In another approach, a fully connected graph is built from human joint estimations, from which the skeleton is estimated using integer linear programming [29]. Another simple method is to add a decoder in the form of multiple transposed convolutional layers after encoding an image to a feature space. Such an application was shown by Xuai *et al.* with a ResNet Encoder [30]. This approach takes cropped human bounding boxes as input data and produces a heatmap for each skeleton joint. This 2D human pose estimation network is used in some experiments in this work (see section 4.2 and moved the post-processing step to extract 2D image positions from heatmaps in the network by using a spatial softargmax in 4.4).

Datasets The more difficult it is to generate annotation data, the fewer data is available. This is also true for 2D human pose estimation datasets, which, compared to millions of labeled data in classification and detection tasks, contain up to hundreds of thousands of annotated human poses, but most often only several thousand. One example dataset is the Leeds Sports Pose (LSP) dataset [31], which contains 2000 hand-labeled images of people performing various sport activities. Another dataset is the 'Frames Labeled In Cinema (FLIC)' dataset [32] composed of around $20,000$ annotated people in Hollywood movies. More data is available in the MPII human pose dataset [33] which contains manually labeled annotations for $40,522$ people doing various activities, collected from YouTube videos. The largest 2D human pose dataset, currently available to the public, is the COCO dataset [25] which provides manually labeled data for over $250,000$ people. All 2D human pose recognition networks, used and developed in this work, use the COCO dataset, at least for pre-training.

Annotation Data For 2D human pose recognition, labels in the form of 2D image positions of human joints of a fixed skeletal structure are required. In addition, the individual joints must be assigned to a specific skeleton.

1.3.4 3D Human Pose Estimation

Three-dimensional body poses are equivalent to two-dimensional body poses, with the difference that they include the additional dimension of depth. The first step in 3D human pose recognition is to define a corresponding 3D space. Estimating the 3D positions directly in the camera coordinate system is unsuitable in most cases since data from different datasets are used for training or validation, and the camera coordinate systems differ accordingly. Therefore, in the presented work, a reference point, the center of the hip, is always assumed as the coordinate system origin. The orientation of the coordinate system may differ from work to work. Besides direct estimation of 3D human joint positions by a deep neural network [34, 35, 36, 37] alternative approaches exist that first estimate 2D human poses followed by 3D human pose regression [38, 39, 40, 41, 42]. Some approaches show the application of model-based approaches [43, 44, 34, 36, 42] to further improve a recognized 3D skeleton. The categorization in bottom-up [35] and top-down approaches [45, 34, 36] is also valid for 3D human pose estimation. Mehta *et al.* show an approach predicting three location-maps for the x, y and z position parameters per body joint [34]. Those location maps encode the distance in x, y, or z direction from the coordinate root (pelvis). The location of a joint on those location maps is retrieved from 2D human pose heatmaps. The results are refined using a kinematic skeleton fitting method. They have also shown how to apply location maps in a bottom-up approach which also handles occlusion better by using redundancy in so-called occlusion-robust pose-maps by representing the decomposed body as torso, limbs, and heads [35]. Luvizon *et al.* show a similar approach to encode depth information in a heatmap but use it only for the depth estimation [37]. It is also possible to directly regress 3D human poses from 2D heatmaps [40] or directly from 2D human pose coordinates [46] which improves when using multiple frames as input [39]. Another approach to retrieve 3D human pose information from 2D human poses is 3D catalog matching [38]. Such approaches rely more heavily on the 2D human pose estimator's output than approaches that also use visual input. Kolotouros *et al.* show how to reconstruct a volumetric model by estimating the parameters for the SMPL statistical body shape model [47] and further improve the model by iteratively fitting on 2D human joints [42]. This work presents an approach similar to [37], but with a straightforward and performant method to retrieve the pose and depth heatmaps in section 4.4.

Datasets The most used datasets in 3D human pose estimation are recorded in motion capture laboratories with calibrated camera sensors. One such dataset is the Human3.6M dataset [48] which contains seven subjects performing 17 different scenarios in a motion capture laboratory while being recorded from four cameras. Another popular dataset is the MPII HumanEva dataset [49] which contains four subjects performing six different actions while being recorded by seven cameras. Another approach was used to create the 3DHP dataset [50] where the authors used a markerless, multi-camera setup to record eight subjects doing eight activity sets with different clothing while being recorded by 14 cameras. They also recorded outdoor scenes using their camera setup and included them in the test set. The 3D Poses in the Wild (3DPW) dataset [51] used an inertial measurement unit (IMU) based approach where the 3D human poses retrieved from the IMUs are aligned with the camera image by using the 2D human pose. They recorded seven actors with different clothing in 60 sequences using moving phone cameras.

Annotation Data For 3D human pose recognition, the x, y, and z position of all human body joints are required. To enable various use cases, they should be provided in camera space and with additional camera calibration (extrinsic and intrinsic) data to transform the positions to any required format. As in 2D human pose estimation, the individual joints must be assigned to a specific skeleton.

1.3.5 Body and Head Orientation Estimation

Body and head orientation estimation approaches are usually handled as separate problems. The estimation of body orientation often originate in pedestrian-related works. Classical approaches regularly used classification of body parts, for example, by using a part descriptor in a sliding window fashion to classify position, scale, and orientation of body parts [52]. Another approach focuses on combining pedestrian orientation and classification by clustering pedestrians in the four categories front, back, left, and right and train classification networks on those clusters, which combined scores serve as the full pedestrian classification [53]. Another approach uses specific detectors for head and body orientation which are converted to a full continuous probability density function and stabilized over time by particle filtering [54, 55]. The authors also discretized the orientation space to 45-degree bins and used a HOG/linSVM based classification system [54]. There is a lot of recent head orientation work using deep learning [56, 57, 58, 59, 60, 61], which usually takes a head bounding box as input and thus is an additional step in the recognition pipeline. Such methods require a head bounding box with a reasonable resolution and thus, high-resolution sensors or people near the camera sensor. Work on body orientation or combined body and head orientation estimation has not yet transitioned well to deep learning approaches. One possible reason could be the lack of appropriate datasets. There are not many standard datasets, and the existing ones are rather small and thus not suitable to train deep neural networks. Heo *et al.* try to overcome this issue on body orientation estimation by using a teacher-student learning framework in which they train a teacher network with labeled data and use this network to generate labels for an unlabeled dataset with which the student network is trained [62]. They have also discretized the output orientation in 45-degree bins, turning the problem into a classification problem [62]. Another work uses CNNs in a random forest that focuses on different body and head parts to recognize the human body and head orientation, with a focus on head orientation [63]. Steinhoff and Göhring propose the usage of IMUs to generate more labeled training data for body and head orientation tasks, but IMU-based approaches are usually hard to sync, suffer from error accumulation, and do not contain global reference points [64]. Wu *et al.* propose the application of 3D human pose estimation approaches (see section 1.3.4) as a basis for body and head pose estimation using a 3D human pose estimation network as a backbone for a classification header which classifies the input in 72 orientation bins [65]. This work proposes a regression approach which is based on full human 3D human pose information, described in section 4.4. Further, it is shown how to train such a network with simulated data to overcome the deficient number of labeled data in this field.

Datasets One available dataset is the HOC dataset [66] which contains $11,000$ full-body images with labels for the front, back, left, and right orientation. Another available dataset is the TUD Multiview Pedestrians dataset [67] containing $5,228$ images which are labeled as one of the eight bins using 45° discretization. In a recent approach Wu *et al.* annotated the COCO dataset with 72 bin discretization for body orientation resulting in $133,380$ labeled images which may be made available on request [65]. In summary, there is a large gap in available datasets for the human body and head orientation estimation, especially when considering full body input data.

Annotation Data Available datasets mostly contain not enough data to train deep neural networks and are usually targeted to classification algorithms using only a low number of classes to discretize the orientation angles. To the authors' knowledge, there is no dataset containing full-body input with 360 degree annotations. There is typically only a label for the yaw angle, not for the pitch, which might be necessary for recognition applications based on the head orientation. Thus, a dataset should contain labels for all 360 degrees for the pitch angle and all 180 degrees for the yaw angle.

1.3.6 Action Recognition

Action recognition describes classifying actions of a human being at a certain time. Examples are 'walking' and 'looking around'. There are various directions of research in the context of human action recognition. Some approaches are based on convolutional neural networks (CNNs). They frequently follow a multi-stream approach [68, 69, 70], which uses an RGB image for visual feature extraction as well as a representation of the temporal flow, usually in the form of optical flow. There is also work which makes use of human poses, either using pose directly [68, 70] or apply some attention like mechanism to get visual features from important areas around the human skeleton [69, 71]. Those approaches often rely on recurrent neural networks [71, 72]. Other approaches rely on handcrafted features extracted from human pose [73, 74]. Labra *et al.* [75] encode 3D joint positions from motion capture data and depth sensors into an image-like data structure. Unlike the EHPI network shown in section 4.2, whole sequences of dynamic length are encoded, which requires interpolation of time frames in color space. At 255×255 pixels, the input data is significantly larger than with the EHPI approach, which increases the computing time for a classification accordingly. The EHPI approach shows that even by using small, fixed time windows, action detection on noisy 2D human pose data from camera-based pose estimation algorithms can achieve high detection accuracy with real-time performance. Most similar to our work in 2D action recognition is the work of Choutas *et al.* [76]. They encode time information in human body joint proposal heatmaps with a color encoding and used this stacked, colored skeleton joint heatmaps as an input for a CNN to classify the action. To reach SOTA performance, they combined this approach with a multi-stream approach [77]. Most of these approaches are relatively complex and therefore do not meet the frame rates required by interactive autonomous systems. The EHPI(2D) approach shown in section 4.2 is much simpler and still delivers competitive performance. A 3D extension of the EHPI2D approach is described in section 4.5.

Datasets There are various action recognition datasets, which usually contain single clips with an assigned action-class label. Examples are the UFC50 [78] and the extension UFC101 [79] containing 50 respectively 101 action classes with about $6,000$ and $13,000$ videos respectively which were collected from YouTube. Another popular dataset composed of commercial and YouTube videos is HMDB51 [80] with its extension JHMDB [81] which also contains joint annotations for pose-based action recognition algorithms. JHMDB was used in experiments in section 4.2. Another database uses Hollywood movies as video source, which are manually annotated with corresponding action labels [82]. Other datasets contain more specialized action categories, for example from sports [83, 84] or cooking [85].

Annotation Data Action annotation algorithms require one or more action labels per frame for supervised learning. There is no general list of human actions available. Thus, the labels depend mostly on the application. There is also often ambiguity between actions, movements, activities, and intentions in the literature. In this work, the term action is considered for elementary actions, such as ‘walking’ and ‘looking around’, which can also occur with other actions. The intention is derived from the actions. For example, the underlying intention of walking and looking around could be the desire to cross the street. As action annotations are usually required for sequences and not only single frames, it is also required to have unique identifiers in the annotation data.

1.3.7 Simulation in Deep Learning

Much work is being done utilizing simulations to create ground truth data for algorithms. Animated 3D models can be used in front of random backgrounds to improve detection algorithms [86], [87]. Another work demonstrates that simulated 3D humans in real environmental backgrounds such as a shopping passage improve the action detection performance of trained models [88]. Shotton *et al.* [89] partially trained the Xbox Kinect-based pose-recognizer on synthetic depth images generated in a 3D human simulation pipeline. However, such a sensor would only work reliably in the near field and under lab conditions. In the field of autonomous driving, much work is done to simulate realistic 3D environments to train and evaluate autonomous car systems [90, 91, 92]. Nevertheless, pedestrians in such environments are mostly just wandering around on predefined paths with low variety in motion and appearance. Souza *et al.* [93] created a database that contains motion capture-driven 3D human models placed inside a virtual environment. They have shown that using image data from such a simulation combined with a small part of annotated real-world data improves action recognition. Khodabandeh *et al.* [94] provide a method to automatically generate an action recognition dataset by partitioning a video into action, subject, and context. Most current work either contains basic representations of humans or represents databases generated procedurally targeting a broad range of motions. Nevertheless, existing work on simulations to train and evaluate algorithms shows a great demand for realistic human motion data. Most current work either contains basic representations of humans or represents bigger databases generated procedurally targeting a broad range of motions. On the other hand, human actions must be simulated very precisely to be significant, for example when it comes to interactions.

Transfer to the Real World A drawback of simulated training data is the transfer of algorithms trained on simulated data to the application on real-world data. There is usually a domain shift, which domain adaptation algorithms should minimize. Such approaches commonly include the use of few real data and a large body of simulated data [95] or methods which use uncorrelated features [96] and advanced domain confusion algorithms [97]. Some approaches try to map from one domain to another or try only to train features that are domain invariant. It is still an open issue which of these approaches best suits the problem of transferring algorithms trained in simulations to a real sensor domain. The experiments show that it is often possible to combine real data with simulated data and transfer data learned from the simulated part to the real-world (see sections 4.1, 4.2 and 4.4). In addition, it is shown how to use pure simulated data with 2D human pose input data as an abstraction level between the camera sensor image and a deep neural network to train an action recognition network which yields near-optimal recognition accuracy on real sensor data in section 4.3.

Chapter 2

Deep Learning

Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.

– LeCun et al. [98, p. 1]

This definition of deep learning states pretty well the basic idea behind it. Thus, input data is processed through various modules, transforming it into an internal feature representation that gets more abstract from the more modules it gets processed. Deep learning is a collective term for neural network architectures which consist of multiple successive modules. One example would be DNNs, which represents neural networks with multiple layers between the input and the output, which enables the representation of complex functions [1, p. 167]. One essential part of this definition is the mentioning of representation-learning methods. In classical machine learning development, data of different types are analyzed by experts. Distinctive features are worked out, and then a reaction to corresponding input data is implemented, for example, by using defined rules or machine learning methods like support vector machines (SVMs) or decision trees (DTs). In representation-learning, features are learned and not engineered like in other machine learning applications [98, p. 1] (*cf.* Figure 2.1, Figure 2.2).

A deep learning system uses these learned internal features to provide a defined output. The output can be, for example, a one hot vector for classification tasks. LeCun *et al.* [98, p. 1] describe the learning part as ‘deep learning discovers intricate structure in large datasets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the presentation in each layer from the representation in the previous layer’. The optimization of the internal parameters is described in section 2.2.3 and the back-propagation algorithm in section 2.2. The process of learning can be divided into three categories.

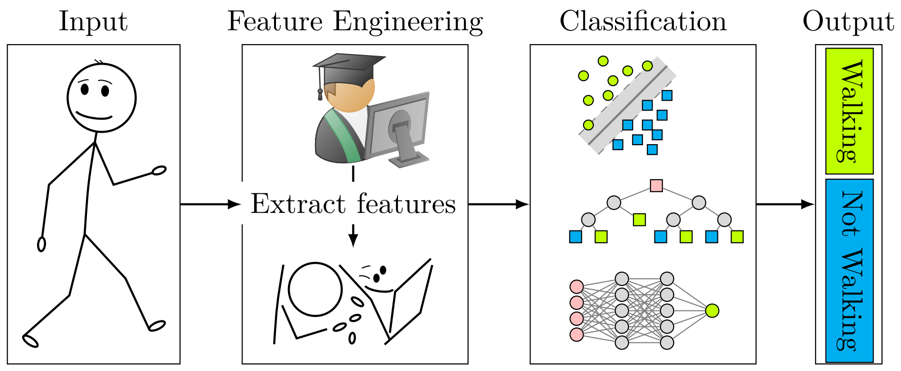


Figure 2.1: Classic machine learning pipeline. An expert manually defines features (for example, a kernel) for a given problem. Those features are used as input data for a classifier (for example, a SVM or a DT) which is trained to provide some kind of output based on the inputted features.

Supervised learning During supervised learning, a machine receives a corresponding output for each input. The machine's task is to learn a function that maps an input to a specific output.

Unsupervised learning During unsupervised learning, only input data is available to the machine. The machine should extract features that describe and categorize the data.

Reinforcement Learning Reinforcement learning can be understood as an indirect form of supervised learning. Instead of a defined output, a reward for outputs is used. The machine learns to output data in a way that maximizes the reward. It, therefore, differs from supervised learning in that no fixed dataset is required.

Since this thesis follows the approach of automatically generating annotated training data by simulation, all learning-based algorithms presented fall into the category of supervised learning.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are networks of interconnected neurons, which are the smallest unit in the network. Neurons are linked to other neurons via connections. The output of one neuron is correspondingly part of the input of another neuron. This link between two neurons is weighted to model the relevance of the value for the receiving neuron. In fully connected neural networks (FCNNs) every neuron in one layer has a connection to every neuron in another layer (see

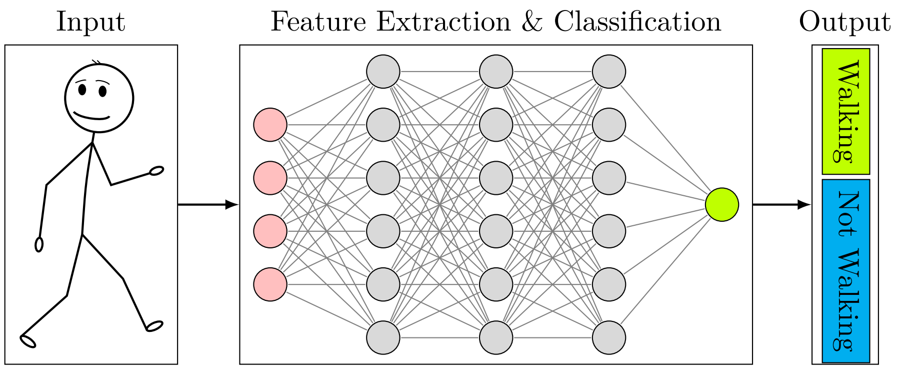


Figure 2.2: Deep learning pipeline. In contrast to the classic machine learning pipeline, the feature engineering is not required. The network extracts features from the input during processing in the hidden layers. In this example, the classification is done based on the features in the last layer.

Figure 2.2). Thus, an input layer $\mathbf{x}$ with length m and an output layer $\mathbf{y}$ with length n result in a weight matrix $W = (w_{ij}) \in \mathbb{R}^{m \times n}$ for $i = 1, ..., m$ and $j = 1, ..., n$ which represents the weights for the combinations of input and output values, correspondingly $\mathbf{y} = \mathbf{x} \cdot W$. A neuron uses the sum of all its inputs in a usually nonlinear activation function $f(\mathbf{x} \cdot \mathbf{w_j} + b)$ (see section 2.1.1) which maps the inputs to a single output. The basic functionality of a neuron is displayed in figure 2.3.

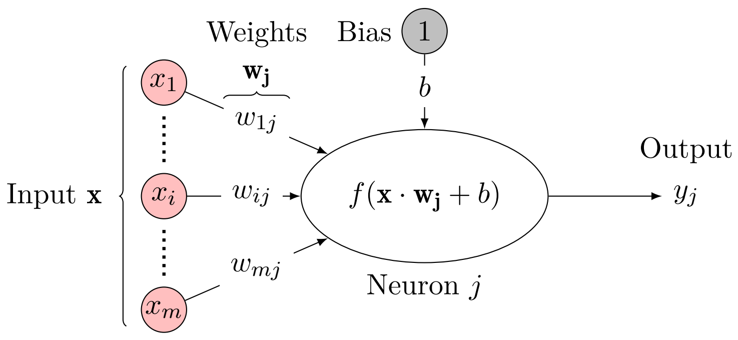


Figure 2.3: Example of an artificial neuron. It uses the sum of weighted inputs $\mathbf{x} \cdot \mathbf{w_j}$ with added bias b as input for a nonlinear activation function f which produces the single output of this neuron, which then can be used as an input of another neuron or as a part of the output of the network.

The bias is a neuron whose input is always one, weighted by variable b. The bias enables the neuron to shift the activation function, thus providing possibilities to reach parameters that otherwise could not be reached. The weights between layers are the parameters of the network which are adapted during the learning process.

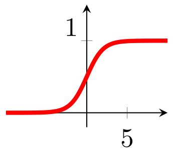
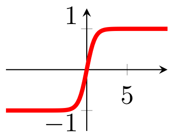
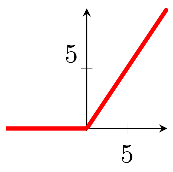
Name	$f(x)$	$f'(x)$	Graph of $f(x)$
Sigmoid	$\frac{1}{1+e^{-x}}$	$f(x)(1-f(x))$	
Hyperbolic tangent	$tanh(x)$	$1-f(x)^2$	
ReLU	$max(x,0)$	$\begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$	

Table 2.1: Example activation functions with their deviations and graphs. The ReLU activation function zeroes all inputs below zero, thus enables sparse representations of inputs. For positive input it behaves linear which also has the benefit of having a simple deviation which is one for $x > 0$ and zero otherwise.

In supervised learning, a loss function (see section 2.2.2) is used that maps the difference between the current output value and the target value from the ground truth to a value. Therefore, it represents the error. The weights of the network should then be adjusted via an optimization function so that the error is minimized (see section 2.2).

2.1.1 Activation Functions

An activation function brings nonlinearity into a neural network and thus enables it to solve complex tasks. Various activation functions exist (see Table 2.1) which are used in different situations. The currently most used activation function is the rectified linear unit (ReLU) [1, p. 174, 98, p. 438] and is seen as one essential part of SOTA feedforward neural networks [99, p. 1026]. Jarrett *et al.* [100] used rectified nonlinearities like $|x|$ and $max(x,0)$ in convolutional neural networks and observed a greatly improve of the performance of recognition systems. This observation was supported by Nair and Hinton [101] which used the $max(0,x)$ nonlinearity to improve restricted boltzmann machines (RBMs). Its application in deep learning was demonstrated by Glorot *et al.* [102], who highlighted the effect of sparsity in addition to the increased performance using $max(0,x)$ as the activation function in deep neural networks.

Most neural networks presented in this work use mainly the ReLU activation function in hidden layers, with exception of LSTMs where the *tanh* activation function is used. The output layers' activation function depends mostly on the use case. For example, a softmax is often applied to the output layer in binary classification tasks to shape the output into a probability distribution. The softmax activation function is defined as $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}}$. It behaves similar to the sigmoid activation function and squeezes values to the range between zero and one. In contrast to the sigmoid function, the outputs of the softmax function are not independent of each other, as the softmax function ensures that all outputs sum to one.

2.1.2 Fully Connected Layer

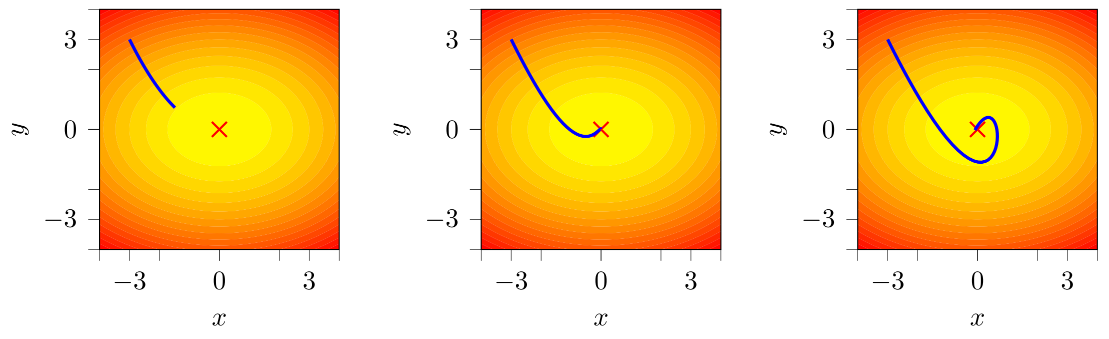
Fully connected layers are primarily used at the end of a network, for example, to build up the last layer in classification tasks so that the set of neurons corresponds to the set of classes. Each neuron of the input layer is connected to each neuron of the output layer via a weighted connection. In Figure 2.2, for example, three fully connected layers are inserted, from which it can be seen that the number of parameters p increases strongly with the number of neurons in the input n and output m as $p = m(n+1)$ with $+1$ for the bias neuron's weight.

2.2 Training

The supervised training of neural networks is an optimization problem. An error is calculated, which is to be minimized by adjusting the weights in the network. One or more loss function(s) (see section 2.2.2) are used to describe the error the network makes for given in- and outputs. Rumelhart *et al.* have introduced the backpropagation algorithm that propagates the error values back through the network to determine the influence of each weight in the network on the total error, thus allowing specific optimization of each parameter. During training, values will first pass through the network from the input to the output layer (forward pass). Afterwards, for each node in the output layer, the influence on the result of each node of the previous layer is calculated. This procedure is repeated by the chain rule for each layer. The network is traversed backward until the input layer is reached, and the error is propagated back through the network accordingly (backward pass). Since the gradients of the output layer determine the influence on the error and the influence of all other nodes is also known, the network can be optimized to minimize the error value. There are different optimization methods, but basically, a value corresponding to the negative gradient is changed by a small amount (see gradient descent in section 2.2.3). The backpropagation algorithm to update the n weights $w_1, ..., w_n$ of a network is shown in equation 2.1.

$$\Delta w_i(t) = -\epsilon \frac{\partial E}{\partial w_i(t)} + \alpha \Delta w_i(t-1) \quad for \quad i = 1, ..., n \tag{2.1}$$

$\frac{\partial E}{\partial w_i(t)}$ represent the gradient calculated from the deltas of the current error rate E with respect to the weight parameter w_i. ϵ represents the learning rate, which controls the rate at which the weights are updated [103, pp. 329-330]. $\Delta w_i(t-1)$ is the result of the calculation from the previous training epoch. α represents the momentum, an inertia term that determines the influence of previous gradients [103, p. 330]. Thus, the previous gradient is included in addition to the current gradient, which is intended to achieve more stable and faster learning. The optimization behavior is visualized by an example for different α values in Figure 2.4. Using gradient descent without the momentum (see figure 2.4a) the x value gets changed so little with each iteration, that the minimum is not reached after 50 iterations. Using $\alpha = 0.8$ (see figure 2.4b) the minimization is stable and reaches the minimum. Using too high values for the momentum is displayed by setting $\alpha = 0.9$ (see figure 2.4c), which leads to overshooting the minimum first before going back towards the minimum.



(a) $\alpha = 0.0$ (b) $\alpha = 0.8$ (c) $\alpha = 0.9$

Figure 2.4: Error surface plot which shows the influence of the momentum on minimizing the error in 50 iterations for the function $f(x, y) = x^2 + 2y^2$ which has its minimum at $x = 0; y = 0$. The blue line visualizes the process of the gradient descent, starting its optimization from $x = -3; y = 3$.

2.2.1 Weight Initialization

To train an ANN, initial values for each parameter in the network are required. Initialization is possible with random values, for example. As shown in 2.2, the training process is strongly dependent on gradients. By a random initialization of the weights from a standard distribution between $[-1, 1]$, it is possible to reach limit values of an activation function (for example, close to -1 or 1 for a *tanh* function), resulting in very large (exploding) or very small (vanishing) gradients. This slows down the training process, worsens the result, or makes the training impossible. To overcome this issue, the weights of all layers must be scaled so that they are neither too small nor too large. If the inputs x and weights w_{ij} of a neural network layer are normalized using standard normal distributions, the result of every product of the element-wise multiplication during a forward pass (see section 2.1) would also have a mean of zero and a standard deviation of one. For n input connections this would result in a layer output x with a mean of zero and a standard deviation of $\sqrt{n}$, which, then, will be used as an input for the next layer. This continues and the deeper a neural network becomes, the higher or lower the gradients become to work with the corresponding inputs. One approach to solve this issue is to scale the layers' weights by $\frac{1}{\sqrt{n}}$ which ensures that the outputs have a mean of zero and standard deviation of one again. Thus, the initial weights for a layer are random variables from a standard distribution between $[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$. However, since activation functions affect the output of a layer in addition to inputs and weights, this approach does not work optimally in practice and can still lead to vanishing or exploding gradients. Therefore, some alternative approaches to initializing the parameters, such as the Xavier initialization [104] for linear activation functions, exist to overcome those issues. They use a distribution as represented in equation 2.2 [104].

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \tag{2.2}$$

U is the uniform distribution in the specified interval. n_j and n_{j+1} represent the number of input and output connections of a layer. They have shown experimentally that this approach results in fast convergence and higher accuracy on the CIFAR-10 image classification task.

He at el. have shown that instead of using $\frac{1}{\sqrt{n}}$ as in linear activation functions, the use of $\sqrt{\frac{2}{n_j}}$ results in a standard deviation of one for the nonlinear ReLU activation function [99] and thus results in faster convergence and better performance when using the ReLU activation function. This kind of weight initialization is also called He initialization.

2.2.2 Loss Functions

A loss function L_i for a sample i is used to express the correctness of outputs of a neural network. The optimization function is used to minimize this loss during training. The loss function is problem-dependent, for example human body joint estimation requires expressing the loss as geometric distances while loss functions in classification tasks usually operate with probabilities. The following loss functions were used in parts of this work.

The mean absolute error (MAE) loss function, also called L1 loss, calculates the absolute difference between the correct output $\hat{y}_i$ and a predicted output y_i as follows:

$$L_i = |\hat{y}_i - y_i| \tag{2.3}$$

Instead of using the absolute value, it is also possible to just use $L_i = \hat{y}_i - y_i$, which is called mean bias error (MBE). Yet, the MBE is usually not used as a loss function because errors of different directions could cancel each other, leaving an uninterpretable error value.

The mean square error (MSE) loss, also called L2 loss, is equal to the MAE but squares the difference between the correct and the predicted output (see eq. 2.4). Due to the squared value, outliers can heavily influence the result of the MSE function. The MSE function is usually used on regression tasks [1, p. 133-134], such as human body joint positions in pose estimation algorithms [27, 30]. In such tasks, it is desirable to punish values further away from the true value.

$$L_i = (\hat{y}_i - y_i)^2 \tag{2.4}$$

The cross entropy loss (CE) loss is usually used in classification tasks [105, p. 236]. For multiclass classification, it is formulated as [105, eq. 4.108]:

$$L_i = -\sum_{c=1}^{C} y_{i,c} log(\hat{y}_{i,c}) \tag{2.5}$$

where c is a class and C is the number of all classes. $y_{i,c}$ is the true label and can be either one or zero. $\hat{y}_{i,c}$ is the predicted output of the network after an applied softmax, and thus, taking the form of a probability distribution. As multiclass classification problems are labeled with one-hot vectors and due to the true label $y_{i,c}$ only the prediction of the true class is considered.

For multi-label classification, the CE loss is adapted to one binary classification problem per class, with the two possible classes C' true and false. The binary form is called binary cross entropy loss (BCE) loss and can be expressed as follows 2.6 [105, eq. 4.90]:

$$L_i = -\sum_{c=1}^{C'=2} y_{i,c} log(\hat{y}_{i,c}) = -\left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\right] \tag{2.6}$$

where $\hat{y}_{i,c}$ is the output of the neural network, but in contrast to the general CE loss with an applied sigmoid instead of a softmax [105, pp. 202-203]. The global loss is just the sum of all BCE losses for all classes in C. This binary form is used, for example, in the action recognition approach described in section 4.2.

2.2.3 Optimization

A basic optimization method is the gradient descent (GD) [106, pp. 536-538, 107, pp. 22-26]. Starting from an initial value (see section 2.2.1), the negative gradient of a parameter update, determined with a loss function over the parameter space (see section 2.2.2), is used as the direction of parameter updates. In this way, it follows the descent of an error curve and thus minimizes the error.

2.2.3.1 Gradient Descent

The gradient descent assumes a differentiable loss function L, which is calculated for each sample i of a dataset with the size N (see equation 2.7).

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L(f(x_i, W), y_i) \tag{2.7}$$

The loss function represents the error between the predicted value of the model $f(x_i, W)$ and the ground truth y_i. This function is derived accordingly to W to calculate the gradients. Then the parameters are updated in the optimization step $t + 1$ by multiplying the gradients with a learning rate $\eta > 0$ (see equation 2.8). This process is repeated until a certain termination condition is fulfilled.

$$W_{t+1} = W_t - \eta_t \frac{1}{N}\sum_{i=1}^{N} \nabla_W L(f(x_i, W_t), y_i) \tag{2.8}$$

2.2.3.2 Stochastic Gradient Descent

The computational effort for the gradient descent, however, scales linearly with the size of a dataset. Thus, it is $O(n)$. Since today's datasets, especially in the field of computer vision, can have several million entries (see section 1.3), the gradient descent is impractical with today's computing power. To avoid this problem, the stochastic gradient descent (SGD) can be used [108, 109]. In this case, an optimization is performed according to each sample in the dataset (see equation 2.9).

$$W_{t+1} = W_t - \eta_t \nabla_W L(f(x_t, W_t), y_t) \tag{2.9}$$

Since there are also outliers among the samples and there is generally noise in the feature space, the approximation of the gradients is also affected by noise. The mini-batch gradient descent (MBGD) is a variant between the (batch) gradient descent and SGD. With this method, a dataset is divided into batches with a fixed size and the weights are optimized after each batch of size M (see equation 2.10).

$$W_{t+1} = W_t - \eta_t \frac{1}{M} \sum_{i=1}^{M} \nabla_W L(f(x_i, W_t), y_i) \tag{2.10}$$

This approach is usually used in practice since it achieves better computing efficiency compared to the gradient descent. In addition, the noise of the gradient approximations is also reduced since averaging the error values is again used for optimization. SGD is often equated with MBGD in general linguistic usage, for example, the PyTorch implementation[1] of SGD also corresponds to MBGD with a batch size > 1.

2.2.4 Batch Normalization

Batch normalization is the normalization of weights for hidden layers, which accelerates the training of a network. The basic idea is to calculate the mean value and the variance of inputs x to a layer in a neural network over a dataset or minibatch and then normalize the inputs before using them in the layer [110]. Thus, a layer would receive $BN(x)$ instead of x where BN stands for batch normalization. The influence of weight changes of previous layers is reduced accordingly, whereby a more stable training can be achieved, especially with deep neural networks, since the deep layers do not have to react so strongly to changes of previous layers [110, p. 450]. This decoupling of the layers should minimize the problem of internal covariant shift, which the authors define as 'the change in the distribution of network activations due to the change in network parameters during training' [110, p. 449].

Let $X = (x^{(1)}...x^{(n)})$ be the set of input values of a layer. For each $x^{(k)} \in X$ the mean $\mu^{(k)}$ (see equation 2.11) and the variance $\sigma^{(k)}$ (see equation 2.12) is calculated by the input values $x_1^{(k)}...x_m^{(k)}$ for mini batch size m resulting in the normalized set of input variables $\hat{X}$.

$$\mu^{(k)} = \frac{1}{m} \sum_{i=1}^{m} x_i^{(k)} \tag{2.11}$$

$$\sigma^{(k)} = \frac{1}{m} \sum_{i=1}^{m} (x - \mu^{(k)})^2 \tag{2.12}$$

For the normalization, a small constant ϵ is added for numerical stability as follows [110, p. 450]:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu^{(k)}}{\sqrt{\epsilon + \sigma^{(k)}}} \tag{2.13}$$

[1]`https://github.com/pytorch/pytorch/blob/master/torch/optim/sgd.py` (accessed on 2022-03-06)

Because the normalization of x would limit the expressiveness of a layer, the authors introduced the learnable parameters γ and β which provide a possibility to scale and shift the normalized values as follows [110, p. 450]:

$$y^k = \gamma^k \hat{x}^k + \beta \tag{2.14}$$

where y^k is the output value after the normalization step, resulting in $BN_{\gamma,\beta}(x)$ as an input for a given layer.

2.2.5 Regularization

The more parameters a neural network has, the more complex functions can be represented by it. This enables the neural network to fit a training dataset very well by using complex functions, including outliers. This behavior is called overfitting and usually leads to a function which is very specific for the given training set, which results in poor performance on data outside the training dataset. To overcome this issue, the network can be regularized [1, p. 248ff]. A usually used way to regularize a neural network is to add a regularization term to the loss function:

$$\widetilde{L}(W) = \frac{1}{N}\sum_{i=1}^{N} L(f(x_i, W), y_i) + \frac{1}{N}\lambda\Omega(W) \tag{2.15}$$

where $\widetilde{L}(W)$ denotes the regularized loss with $\{\lambda \in \mathbb{R} \,|\, 0 \leq \lambda \leq 1\}$ as a hyperparameter for the regularization term $\Omega(W)$, thus setting $\lambda = 0$ results in $\widetilde{L}(W) = L(W)$ [1, p. 230]. The effect on the training is demonstrated in Fig. 2.5. The unconstrained network overfits to the training dataset, resulting in a rather complex function (see Figure 2.5a). A heavy regularization of the network leads to a too simple function that cannot approximate the optimal function (see Figure 2.5b). A slightly regularized network leads to a function better representing the distribution of the samples from the training dataset (see Figure 2.5c).

L_2 **Parameter Regularization** One common regularization term is the L_2 regularization $\Omega(W) = \frac{1}{2}||W||_2^2$ [1, p. 231] also known as weight decay. This regularization term penalizes large weights, which should simplify the model.

Dropout One regularization technique used especially often in computer vision [21, 10, 9] is dropout. Using dropout means that in each optimization step, a portion of the neurons of one or more layers is removed randomly [111, p. 1930]. The main idea behind this removal of neurons is to force the neural network to consider multiple features to achieve its optimization goal rather than relying heavily on some robust features [111, p. 1932]. It results in a reduction of high weight

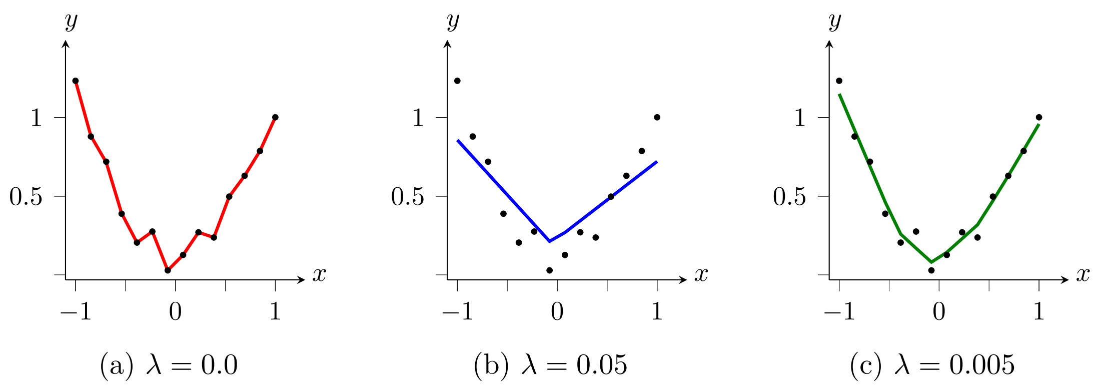


Figure 2.5: Visualization of the influence of weight decay as a regularization term. 20 data points are sampled from $y = x^2 + \xi$ where $\{x \in \mathbb{R} \,|\, -1 \leq x \leq 1\}$ and $\{\xi \in \mathbb{R} \,|\, 0 \leq \alpha \leq 0.3\}$. ξ is a random variable to add noise to the parabola. A neural network was used to approximate the original function sampled from with noise. The used neural network contained two fully connected layers with about 6000 trainable parameters and is regularized using weight decay with different λ values.

values. Dropout can be applied on each layer, but in most cases, it is applied on layers with many neurons, which are mostly responsible for overfitting.

Batch Normalization Batch normalization is not directly a regularization method, rather than a technique to accelerate the training process (see section 2.2.4). Nevertheless, it leads to a slight regularization of neural networks, which experimental studies suggest [110, p. 454]. This slight regularization effect can be enough to make dropout unnecessary [1, p. 268].

Early Stopping Another method to regularize a neural network is early stopping. The idea behind early stopping is to observe the validation error during the training, and if the validation error starts to increase, the training is stopped [105, p. 259-260]. In practice, the model parameters are saved when the validation error decreases. If for a predefined number of iterations no reduced validation error is observed, it is concluded that the network is overfitting, and the training has to be stopped.

Data Augmentation As Goodfellow *et al.* state, 'The best way to make a machine learning model generalize better is to train it on more data. [1, p. 240]'. Part of this work is the rapid generation of data via simulation, with which networks should be regularized better. An additional approach to increase available data is the application of data augmentation techniques. The main idea is to apply transformations on available data to alternate the appearance. In computer vision, rotations, scaling, or changes in color are often used.

2.2.6 Convolutional Neural Networks

A CNN is a neural network that contains one or more convolutional layers. Most neural networks in this work are CNNs. This section gives an overview of the layers of which the CNNs in this work are composed. Those building blocks are combined with techniques highlighted previously, such as activation functions (see section 2.1.1) and batch normalization (see section 2.2.4).

2.2.6.1 Convolutional Layer

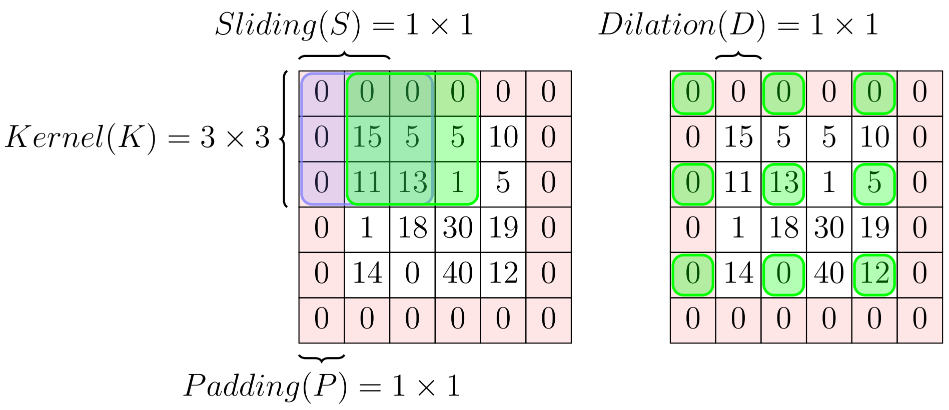


Figure 2.6: Overview of common convolutional layer parameters. The kernel is highlighted in green. The padding pads the actual matrix content with n constant values. This provides control over the output size in the subsequent layer. Sliding defines a constant value by which the kernel window is moved through the input data. The left example highlights the old position of the kernel in blue and the new position after moving 1 step to the right in green. Dilation defines the space between the kernel rows and columns. The kernel size defines the size of the kernel, which is moved over the input data.

Most networks in this work use convolutional layers, which learn the kernel parameters. A kernel has a size $K = m \times n$ and is applied in a sliding window manner. The sliding parameter S controls the movement of the sliding kernel in the horizontal and vertical directions. The application of such a kernel usually leads to a reduction of the input size. It is possible to overcome this by adding padding around the input data, which is defined by parameter P. The usual method is to use zero padding, but it is possible to use other approaches, such as copying adjacent input data. It is also possible to control the horizontal and vertical distance of the sliding as well as the kernels' shape by skipping lines or columns, which is called dilation, which parameter D specifies the distance between kernel cells. An overview of the parameters with which a kernel's behavior can be controlled is given in Figure 2.6. The upper case parameters in all equations in this section refer to symmetric parameters to prevent separate formulas for both width and height. The output size can be determined using these parameters as follows:

$$O = \frac{I - K + D(K-1) + 2P}{S} + 1 \tag{2.16}$$

where O is the output size, I the input size, K the kernel size, D the dilation size, P the padding size and S the sliding step size and may be calculated separately if different values are used for the height and width of the specific parameter.

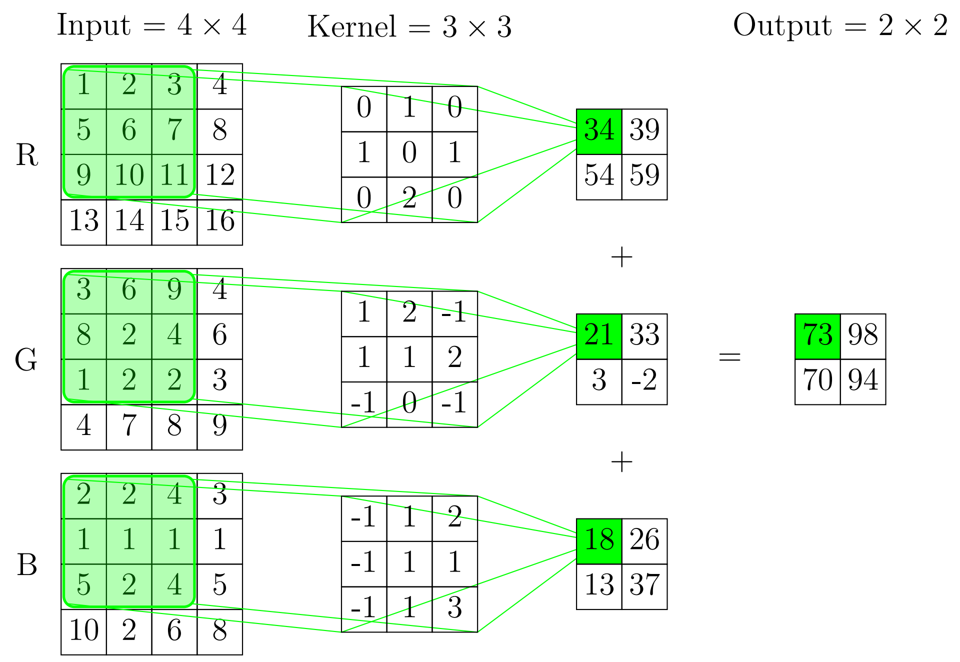


Figure 2.7: Example of the application of kernels in CNNs on RGB input data with one output channel. The example shows an input image of size $I = 4 \times 4$ with a padding size of $P = 0 \times 0$ and a sliding of $S = 1 \times 1$ and a kernel size of $K = 3 \times 3$ without dilation resulting in the output size $O = 2 \times 2$. Highlighted in green is the first filter applied on all channels, as well as the calculation of the one output channel.

As shown in Figure 2.7, the actual calculations of the output of a convolution are just a combination of products and sums. For each channel, there is a kernel with different parameters. This kernel is moved stepwise (S) over the input data. Every input value in the current window is multiplied by the kernel parameter at the specific position. Afterwards, all values in the window are summed up, and the sum is written to the corresponding continuous position in the output. The final result is composed of all the data of all channels, where the individual values are added according to their position. The kernel parameters are shared across all locations of the input and reapplied through the sliding window approach. As such, a CNN contains fewer parameters than FCNNs. The actual number of kernels can be derived from the number of input channels C_i and output channels C_o as described in equation 2.17.

$$N_k = C_i C_o \tag{2.17}$$

2.2.6.2 Transposed Convolutional Layer

Another layer that is frequently used in the context of this work is the transposed convolutional layer. This is often referred to as a deconvolutional layer, which is incorrect because it does not perform deconvolution but also convolutions, but it performs upsampling instead of downsampling. This process is shown exemplary in Figure 2.8.

Transposed convolution also uses one kernel per input-output channel combination. In contrast to convolutions, however, each input value I_i is multiplied by each element in the kernel K_j so that the output is scaled up accordingly.

In contrast to this more intuitive representation of transposed convolutions, Figure 2.9 shows how transposed convolutions are actually implemented as convolutions.

A padding of $K - 1$ is added to the input. Afterward, the input is convolved as in standard convolutions. Note that the parameters used in convolutions need to be handled differently in transposed convolutions. First of all, the kernels must be flipped vertically and horizontally (*cf.* Figure 2.8 and 2.9). The padding parameter is used to reduce the actual padding around the input. For example, in PyTorch, the padding is implemented as $D(K - 1) - P$ with defaults $D = 1$ and $P = 0$, thus the padding parameter P reduces the padding size which then leads to a reduced output size[2]. The stride parameter also differs in its behavior. When using the view in figure 2.9, the stride parameter adds $S - 1$ spaces between the values of the input data, behaving more like the dilation parameter in standard convolutions. Using stride values, $S > 1$ may result in ambiguities in the output size. For example, a convolution from 5×5 and 6×6 inputs with $P = 0$ and $S = 2$ both lead to the output size of 2×2. This also applies to the transpose, which will usually generate an output of size 5×5 given an input of size 2×2. Suppose a network

[2] `https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html` (accessed on 2022-03-06)

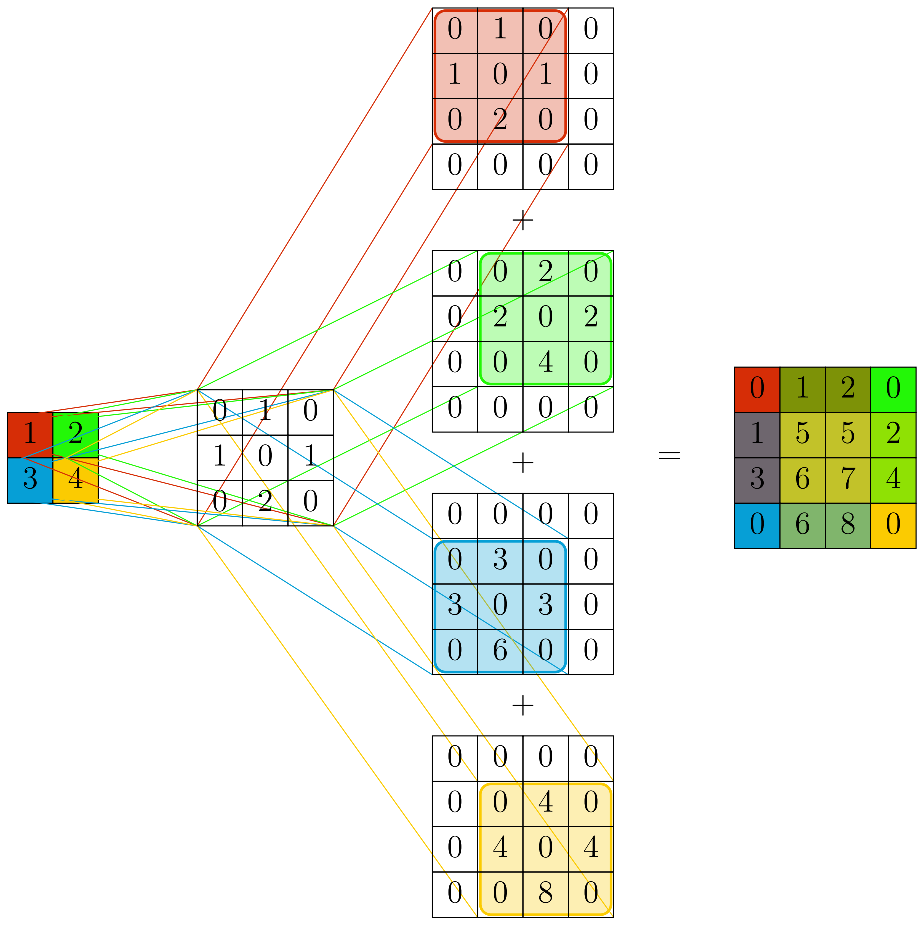


Figure 2.8: Example of a transposed convolution. Note that the stride of $S = 1 \times 1$ is applied on the output not the input by defining the step size of which the results, after applying the kernel, are combined. Thus, S directly influences the size of the output. For example, $S = 2 \times 2$ would result in an output size of $O = 5 \times 5$. The colored areas show the result of the transposed convolution applied on one input value, for example the red 1 in the input results in the red 3×3 area. The colors in the final result are mixed according to their overlap (*cf.* the red 0 in the upper left without overlaps and the 1 below with an overlap between red and blue).

design requires symmetric behavior of convolution and transposed convolution an additional parameter P_o where $P_o < S$ is required, which defines if there should be one-sided padding added to the output. As such, the size of the output of a transposed convolution can be calculated as shown in equation 2.18.

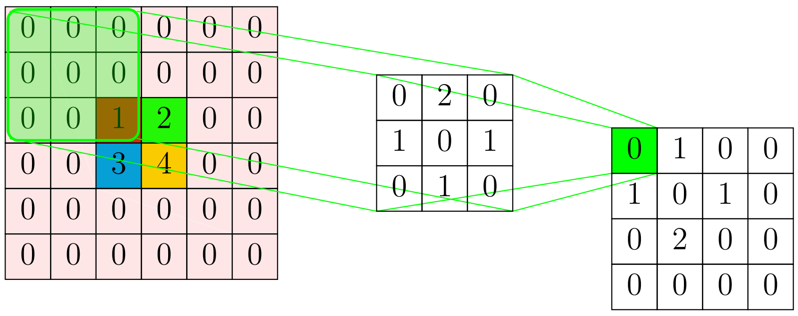


Figure 2.9: Example of a transposed convolution in an alternative view, which shows how convolutions are used.

$$O = S(I - 1) + (K - 1) - 2P + P_o + 1 \tag{2.18}$$

Transposed convolutions are mostly used to scale up a feature map, for example, to display a heatmap, segmentation, or depth map. In this work, 2D and 3D human poses as well as orientation heatmaps are generated using transposed convolutions (see sections 4.2 and 4.4).

2.2.6.3 Pooling Layer

Pooling layers are another method to downsample input data. The input data is divided into a grid shape using a filter parameter of size $m \times n$ and stride S. A corresponding pooling operation is applied to each filter. Examples are *max pooling*, where the maximum value in the filter range is used, or *average pooling*, where the average value is used. There are other pooling methods, such as *L2 norm pooling*, but the neural networks used in this work apply max pooling or average pooling exclusively.

2.3 Metrics

Depending on the learning task, various metrics are used for validation, testing, and within loss functions. In the following, metrics are presented that are used in the context of this work.

2.3.1 Classification

The classification task can be divided in the following subtasks [112, p. 428].

Binary An input will be classified into one of two non-overlapping classes C_1 or C_2.

		Ground Truth	
		P	N
Prediction	P	TP	FP
	N	FN	TN

Table 2.2: Binary Confusion Matrix

Multi-class The input will be classified into one of n non-overlapping classes C_1 to C_n.

Multi-labeled The input will be classified into several of n non-overlapping Classes C_1 to C_n.

Hierarchical The input will be classified into **one** class, the class labels are structured in a hierarchical way.

Binary, multi-class, and multi-labeled classification tasks are used in parts of this work. The following parameters characterize a classification tasks:

Total number of samples (NS) The number of all samples.

True positives (TP) are the number of positive samples recognized correctly.

True negatives (TN) are the number of negative samples recognized correctly.

False positives (FP) are the number of positive samples not recognized.

False negatives (FN) are the number of negative samples recognized as positive samples.

These parameters can also be represented together in a binary confusion matrix (see Table 2.2).

It is also possible to display classification results of multiple classes in a confusion matrix. This is shown in Figure 2.10 using the example of a three-class classification.

There are various metrics that can be used to quantify the quality of a classification algorithm. Most of these metrics should not be used alone, but in combination. Otherwise, not all aspects of the results are covered. For example, a 100% true positive rate does not mean anything, as this can also be achieved with a 'classifier' that simply always returns the same label.

2.3.1.1 Accuracy

The accuracy metric simply describes the correct predictions in relation to the total number of samples and is calculated as shown in equation 2.19.

$$\text{Accuracy} = \frac{TP + TN}{NS} \tag{2.19}$$

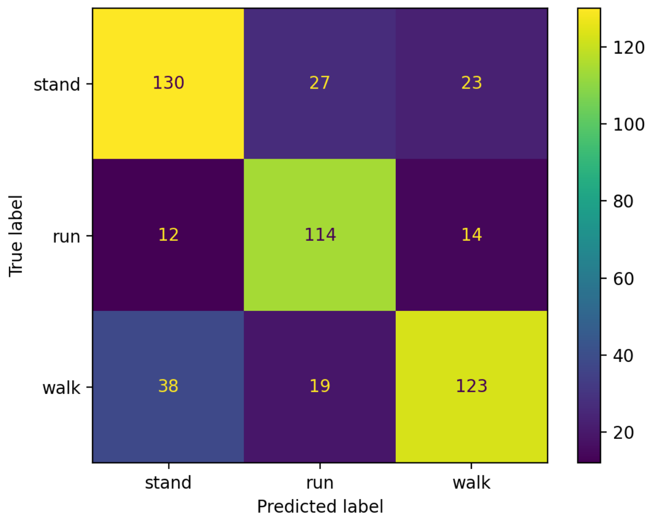


Figure 2.10: Example of an unnormalized three class confusion matrix.

The problem with accuracy alone is that it may be misleading, especially in uneven datasets. For example, a dataset with 10 true positives and 90 true negatives may lead to 90% accuracy if a classifier vote all samples negative. Nevertheless, the classifier would be useless. This information cannot be retrieved from the accuracy value only. Therefore, precision and recall metrics can be used in addition to accuracy.

2.3.1.2 Precision

The precision metric describes the relation between true positives and the total number of positively labeled samples (see equation 2.20).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.20}$$

Thus, the precision describes how relevant the positive label is. Since the precision value excludes false negatives, it is usually used together with the recall.

2.3.1.3 Recall

The recall is also called the true positive rate and describes the true positives in relation to the false negatives (see equation 2.21), thus describing the effectiveness of a classifier to identify positive labels [112].

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.21}$$

2.3.1.4 Specificity

The contrast to recall is the specificity metric, which describes the relation of true negatives to false positives (see equation 2.22) and thus how effectively a classifier can identify negative labels [112]. It is also called true negative rate.

$$\text{Specificity} = \frac{TN}{FP + TN} \tag{2.22}$$

2.3.1.5 F_1 Score

Closely related to recall and precision is the F_1 score, which is the harmonic mean of precision and recall (see equation 2.23) [112].

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.23}$$

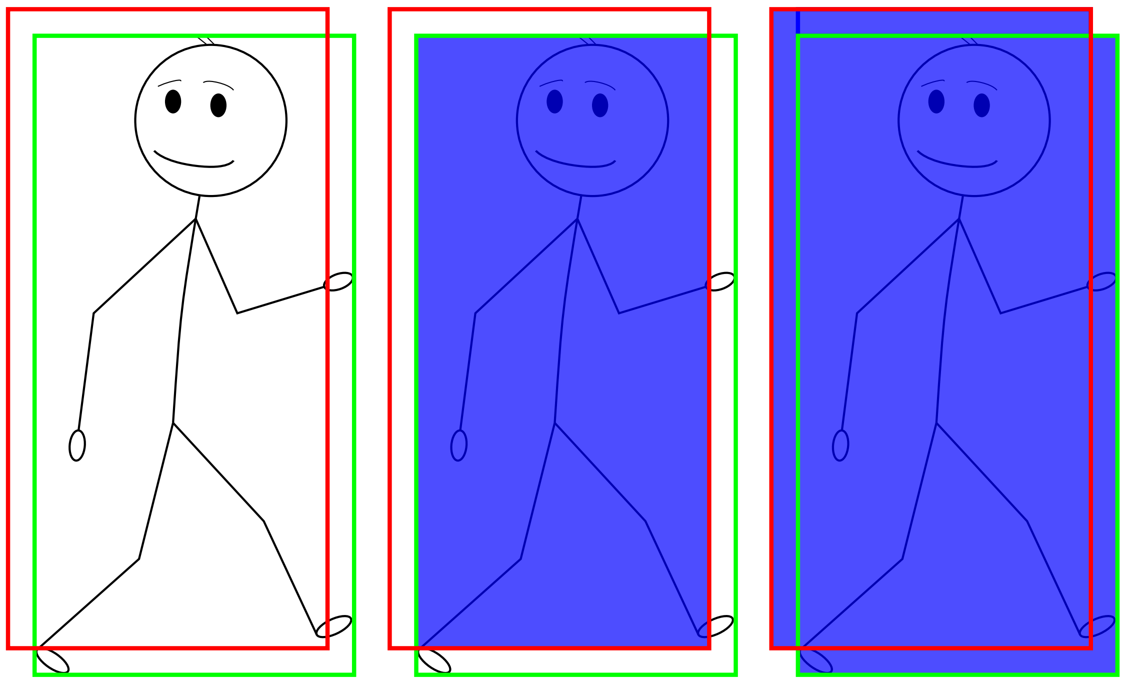
2.3.1.6 Multi-label Classification Metrics

For multi-label classification, the following metrics are mostly used:

$$OP = \frac{TP}{TP + FP} \tag{2.24}$$
$$CP = \frac{1}{C}\sum_i = 1^C \frac{TP_i}{TP_i + FP_i} \tag{2.25}$$
$$OR = \frac{TP}{TP + FN} \tag{2.26}$$
$$CR = \frac{1}{C}\sum_i = 1^C \frac{TP_i}{TP_i + FN_i} \tag{2.27}$$
$$OF_1 = 2 \times \frac{OP \times OR}{OP + OR} \tag{2.28}$$
$$CF_1 = 2 \times \frac{CP \times CR}{CP + CR} \tag{2.29}$$

where P is precision, R is recall and F_1 the F_1 score. $O(...)$ describes accordingly the F_1, precision and recall metrics over all classes and $C(...)$ the mean of the metrics per class. It should be noted that C(...) gives a distorted representation for unbalanced datasets. On such unbalanced datasets, the balanced accuracy (BA) can also deliver valuable information and is defined as:

$$BA = \frac{\text{recall} + \text{specificity}}{2} \tag{2.30}$$



(a) bounding boxes (b) Area of overlap (c) Area of union

Figure 2.11: Bounding box IoU metrics visualized. The green bounding box is the ground truth and the red bounding box the prediction.

2.3.2 Detection

An example for a detection task is shown in Figure 2.11a.

The example shows a pedestrian as well as a predicted and a ground truth bounding box. A metric for detection tasks should contain a measurement for the correct position and scale of the bounding box and a measurement for the predicted class in multi-class detection tasks. To measure the quality of the predicted bounding box, usually, the intersection over union (IoU) is used. Let A_{gt} and A_{pred} be the area of the ground truth and predicted bounding boxes as well as A_{inter} the area of overlap, the IoU is calculated as follows:

$$IoU = \frac{A_{\text{inter}}}{A_{\text{gt}} + A_{\text{pred}} - A_{\text{inter}}} \tag{2.31}$$

The IoU is the ratio of the area of overlap of a ground truth bounding box and the predicted bounding box (see Figure 2.11b) against the area of union (see Figure 2.11c). The IoU is often used in loss functions. However, research results are usually reported as the mean average precision (mAP), which uses the IoU to determine if a bounding box is a valid bounding box and checks the correct class labeling afterward.

2.3.2.1 Average Precision (Detection)

The average precision (AP) represents the average precision value in respect to recall values between zero and one [113, pp.158-161]. To calculate the average precision value, first, the detected objects are transformed in classification results by using an IoU threshold in combination with the classification label to decide if the detection is a true positive (TP). Afterward, the results are ranked, for example, by using the confidence score. Next, for all samples, the precision, and recall value is calculated. The precision value always takes all samples up to the current rank into account. Thus, as the recall value increases, the precision decreases. This mechanic may be used in applications to either prefer true positives with a low number of false positives (high precision) or to include a low number of false negatives at the cost of more false positives (high recall). An example is plotted in Figure 2.12, such a plot is called precision-recall curve [113, p. 158]).

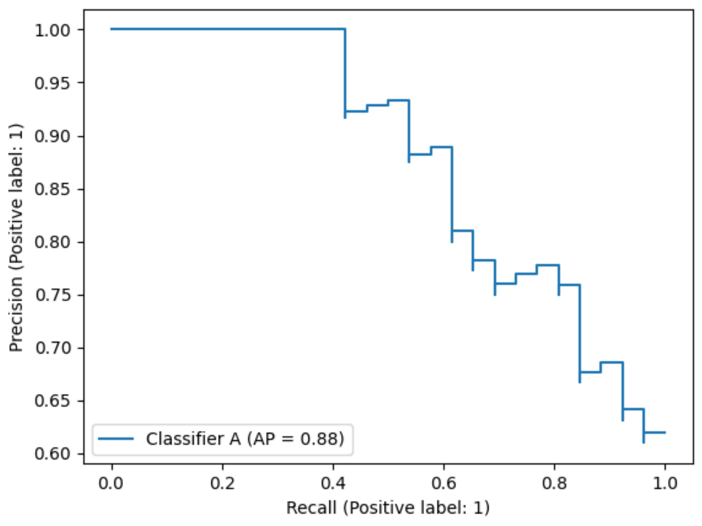


Figure 2.12: Example of a precision-recall curve.

The precision values in the graph are usually interpolated as p_{interp} for a certain recall value r which is defined as [113, p. 159]:

$$p_{\text{interp}}(r) = \max_{r' \geq r} p(r') \tag{2.32}$$

The AP metric represents the area under this graph. The value is usually interpolated by sampling the recall values, to overcome the computational expense of calculating the integral (see equation 2.33).

$$AP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, ..., 1.0\}} p_{\text{interp}}(r) \tag{2.33}$$

where 11 corresponds to the traditional 11-point interpolated average precision which samples the recall levels $0.0, 0.1, 0.2, ..., 1.0$ [113, p. 159]. For multiclass detection, usually the mAP is reported. For a set of C classes, the mAP is calculated as follows:

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^{C} AP(c) \tag{2.34}$$

2.3.3 2D Human Pose Estimation

2.3.3.1 Percentage of Correct Keypoints (PCK)

The percentage of correct keypoints (PCK) metric introduced by Yang and Ramanan [114, p. 2884] evaluates how well individual human joints are detected by an algorithm. It defines a joint to be correctly detected when it falls within $\alpha \cdot \max(h, w)$ pixels of the ground truth joints' position, where h and w denote the height and width of the bounding box of a person and α the relative threshold for considering the correctness of the joints position. Equation 2.35 highlights the calculation of the PCK metric for a skeleton with N joints and Euclidean distances d_i between the predicted and the ground truth joint positions.

$$PCK = \frac{\sum_{i=1}^{N} TP_i}{N} \text{ with } TP_i = \begin{cases} 1, & \text{if } d_i < \alpha \cdot \max(h, w) \\ 0, & \text{otherwise} \end{cases} \tag{2.35}$$

There is also a variant of the PCK metric called PCKh, which uses the head bounding box instead of the full-body bounding box as a threshold reference size to overcome the influence of articulation [33, p. 3689-3690]. The PCK score is usually reported together with the used threshold value, for example, PCK@0.2 denotes the PCK value with a threshold size of 20% of the bounding boxes' largest side [114, p. 2884].

The PCK metric is thus heavily dependent on the bounding box size, the chosen threshold, as well as the ground truth labels. Figure 2.13 highlights how this might influence the expressiveness of the PCK metric. Except for the right elbow, the pose estimation algorithm delivers a fairly accurate pose from a human perspective, but the PCK@0.05 score is only at 36%. The problem is that the labeling of 2D human joints is not a task with a single correct solution, since the individual joints cannot be assigned exactly to one pixel. For example, the position of the hip joints in Figure 2.13a were labeled rather on the top outside, but the predictor in Figure 2.13b rather outputs the center hip position. Both can be correct, but this is not reflected in the PCK score. One could increase the threshold to ignore these labeling inaccuracies. However, the higher the threshold, the higher the risk that incorrectly labeled joints, such as the right elbow in the example, will be recognized as correct predictions.

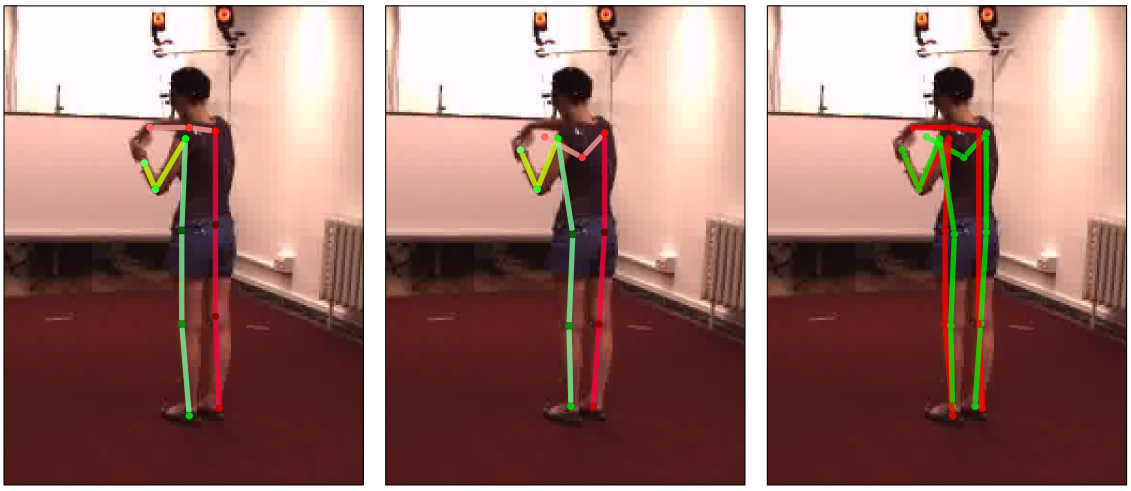

(a) GT (b) Prediction (c) Comparison

Figure 2.13: 2D human pose estimation example with an PCK@0.05 score of 36%.

2.3.3.2 Percentage of Detected Joints (PDJ)

The percentage of detected joints (PDJ) metric is similar to the PCK metric, but uses the bounding box diameter $\sqrt{w^2+h^2}$ instead of the largest side $max(w,h)$ of the bounding box as reference value [115, p. 5]. The original paper used the torso diameter, which leads to problems with people standing sideways. Thus, the full bounding box diameter is usually used, which is heavily influenced by a person's articulation.

$$PDJ = \frac{\sum_{i=1}^{N} TP_i}{N} \text{ with } TP_i = \begin{cases} 1, & \text{if } d_i < \alpha\sqrt{w^2+h^2} \\ 0, & \text{otherwise} \end{cases} \tag{2.36}$$

Equation 2.36 shows the calculation of the PDJ metric for a skeleton with N human joints and Euclidean distances d_i between the predicted and the ground truth joint positions.

2.3.3.3 Object Keypoint Similarity (OKS)

The object keypoint similarity (OKS) score is another approach to define a metric for joint distances. It is mainly used in the evaluation of the COCO dataset benchmarks[3]. The main difference to the PCK or PDJ metric is the usage of a joint-specific, constant factor k, which controls the influence of individual human joints. For example, joints in the face are weighted less than body joints. They also use a scale value s to normalize the joint distances based on the human scale. This scale is usually derived from the ground truth segmentation area or, if this is not available, from the object's bounding box area[4]. Equation 2.37 shows the OKS metric for one skeleton with N joints and the Euclidean distances d between the ground truth and the predicted joints.

$$OKS = \sum_{i=1}^{N} \exp\left(-\frac{d_i^2}{2s^2k_i^2}\right) \tag{2.37}$$

2.3.3.4 Average Precision for Keypoint Estimation

As in detection metrics, results in pose estimation algorithms are also often reported using the AP metric. Instead of the IoU in detection metrics, a joint distance score, for example, the OKS score, is used to determine TP human joints. The threshold for accepted joints is usually reported as $AP^{threshold}$. For example, in the COCO dataset benchmark, usually AP^{50} and AP^{75} are used. The COCO dataset benchmark also uses separate evaluations of small, medium, and large bounding boxes, which are denoted as AP^{S}, AP^{M}, and AP^{L}.

2.3.4 3D Human Pose Estimation

3D human pose estimation is similar to the 2D human pose estimation in its evaluation. What must be taken care of in 3D human pose estimation is the 3D coordinate system that is used. In most available datasets and benchmarks, the metrics are applied in a coordinate system that has its origin at the person's body to be recognized. All joint positions are then specified relative to this point. The pelvis is most commonly used for this purpose.

2.3.4.1 PCK3D

The PCK metric can also be applied to 3D human pose estimation. It uses the Euclidean distance of the ground truth and predicted human joints in 3D space. Yet, it differs in the threshold retrieval, which is based on the scale of a person in 2D human pose estimation. The to be predicted 3D body size is independent of the distance to the camera. Thus, it depends solely on the actual size of a human. As

[3] `https://cocodataset.org/\#keypoints-eval` (accessed on 2022-03-06)

[4] `https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/coco.py` (accessed on 2022-03-06)

such, the threshold for PCK3D is usually set to 150mm, which roughly corresponds to half of the head size.

2.3.4.2 Mean Per Joint Position Error (MPJPE)

A metric that is often used in 3D human pose estimation is the mean per joint position error (MPJPE) metric, which reports the Euclidean distance between the ground truth and predicted joint position. As the distance to the camera does not influence the skeleton size, the only influencing factor besides the quality of the estimator is the actual human size. A large enough dataset, with a considerable variety of human body sizes in the test set, should average out this factor.

Chapter 3

Simulation

The simulation must provide ways to generate automatically annotated sensor data. The focus of this work is on RGB image data. Therefore, only the generation of RGB camera sensor data is addressed. The principles of the generation of data can be extended to other sensors. One example would be LiDAR sensor data, which could be simulated using ray casting [116, p. 1]. Since this work is aimed at human detection, the focus of the simulation is likewise put on the simulation of human avatars. The basic prerequisites to simulate a human being are human motion, human shape, and an environment. In general, it is desirable to create a high data variability to achieve better performance of data-driven algorithms. High variability is significant in human movements, as movement executions can vary from person to person and in different situations. Increased variability can also be reached by placing virtual sensors that observe people from various perspectives. This free placement of sensors in the virtual environment is one of the biggest advantages of a simulation. Different perspectives offer many variations in the representation and help to achieve a better generalization. In addition, sensors can be easily rearranged based on requirements to obtain more sensor information. Examples would be sensor positions in different locations of different vehicles approaching a simulated person from different angles.

3.1 General Requirements

The requirements for the simulation environment can be divided into three areas. On the one hand, the user should be supported in obtaining the required raw data, such as the animations and the 3D avatars, by automating laborious or error-prone tasks. On the other hand, there are requirements for the generation of simulation content, for example, the actual sensor data. Last, there are requirements for the labels to be generated.

3.1.1 Required Data in the Simulation Framework

The most important part of the simulation framework is the data that the simulation environment should use to generate sensor data. This part also requires the most manual effort to obtain the data, prepare it for the simulation, and annotate it. The user should be supported by automating complex and error-prone tasks. For example, the rigging of 3D scanned models should be automated. Furthermore, many different movements should be recorded, which means a high effort in a motion capture lab to instruct participants, perform recordings, label recordings, and prepare the simulation of data. This process should be supported to minimize the manual effort.

3.1.2 Data Generation

Sensor data should be generated as automatically as possible within the simulation environment. Thus, one of the main requirements is to reduce manual intervention as much as possible. A user must be able to load raw data such as animations and 3D characters into the simulation and use them directly with each other without having to make manual adjustments. The simulation should create a high variability in the generated sensor data by automatically generating different movements with different human 3D models. The simulation must do this independently and generate the data without further human intervention. This should also allow the generation of many data that takes a long time. The user must be able to specify which raw data to use before a simulation run. For this purpose, there must be filtering options, such as included actions, size of avatars, or the age of persons from whom a movement originates. Furthermore, the user should also receive support in the placement of sensors for cases in which as many perspectives as possible are required, but the sensors do not have to be placed in special positions, such as in a vehicle windshield.

3.1.3 Ground Truth Labels

The requirements for the ground truth to be generated can be derived from the tasks to be fulfilled as well as the metrics used (see section 2.3). There are many use cases and correspondingly different requirements for labels. In this work, the focus is on classification (for example, actions), detection (pedestrians), 2D and 3D human pose recognition, as well as orientation estimation. Thus, the main requirement on labels to be generated is classification labels, 2D and 3D joint positions as well as the orientation of the body and the head. However, information about all elements is available in the simulation environment, which can also be used to generate other types of labels. Areas like variation of environments, crowd simulation, or weather simulation affect the generated sensor information and the simulated individuals.

However, the inclusion of these elements would be beyond the scope of this work.

3.1.3.1 Classification

Every object in the simulation must be identifiable. For each object in the simulation, it must be specified to which class it belongs. In addition, each object must be uniquely identifiable to differentiate between objects within the same class.

3.1.3.2 Detection

The simulation must provide information about the position and rotation of objects relative to the observer. 2D and 3D bounding boxes must be made available for this purpose.

3.1.3.3 2D and 3D Human Pose Estimation

The simulation should provide the 2D and 3D positional information of joints of the human body. Using these joints, the skeleton structure of the human can be reconstructed, and the exact execution of an action is represented. For 2D human poses, the joint positions should be output as x and y image coordinates. Furthermore, a skeleton structure must be selected covering various skeleton formats from benchmarks such as COCO [25] or Human3.6M [48]. For 3D human poses, x, y, and z coordinates shall be provided in 3D space. These must be provided either for each sensor coordinate system or in global form, with additional information about the sensor's intrinsic and extrinsic camera calibration.

3.1.3.4 Action Recognition

For tasks such as action recognition, it is important that generated data can be temporally associated. To provide temporal context, every sensor information must be annotated with a timestamp. This global timestamp can be used to infer local temporal context. For example, a person flinches after another person screamed. Furthermore, it is necessary to specify the period of time over which the actions are performed.

3.1.3.5 Semantic Segmentation

Semantic Segmentation tasks are not directly part of this work. However, for example, the OKS metric derives the scaling of a person from segmentation data. Therefore, the simulation should also be able to segment objects and especially persons in a camera sensor image.

3.1.3.6 Depth

Another type of annotation that is not directly addressed in this thesis, but is challenging to annotate from real monocular camera data correctly, is depth data. In a 3D simulation (see section 3.2), these are already generated during the rendering process and can therefore be output quickly, but are not used for machine learning tasks in this work.

3.2 Simulation Framework

The simulation framework is visualized in Figure 3.1 and can be used to generate annotated data to train data-driven algorithms. The simulation framework requires motion capture data as well as 3D human models to generate simulation data. The left half of Figure 3.1 shows the data acquisition part and describes that participants are recorded with real-world sensors, like a motion capture system to capture movements or a 3D scanner to generate 3D avatars. Some of these recorded data already provide information that can later be used as ground truth labels. For example, information such as the age or weight of a person who has been motion captured and scanned in 3D can be stored and later output in addition to simulated sensor data. In the simulation, the collected motion data is transferred to 3D human models, which then move accordingly in the 3D environment. We place virtual camera sensors in this 3D environment that record the 3D models as they move. Since every aspect of the simulation is known, information is available for each virtual sensor that can be stored as a ground truth label. Examples are human joint locations, object locations, as well as the current action of a person.

Two crucial aspects of the proposed framework are the ability to modify the environment and the content within with fully known and observable elements, as well as having access to metadata. Metadata has two primary purposes. The first purpose is to make relevant data available in simulations. Knowing a participant's gender, age, race, height, and weight enables the selective querying of models based on these parameters. For example, it can be used to create a simulation that only contains models with the visual characteristics of people over 60 years of age. The second purpose of metadata is its usage in the output of the simulation. Metadata of simulated components can be used directly as additional ground truth for the training and validation of machine learning algorithms. It is ground truth that usually cannot be derived directly from a sensor output but has to be interpreted from visual stimuli, context information, and background knowledge.

The key benefits of using a simulation for the use-cases in this work are:

1. To generate training data, only 3D human models, motion capture data, and an environment are required.

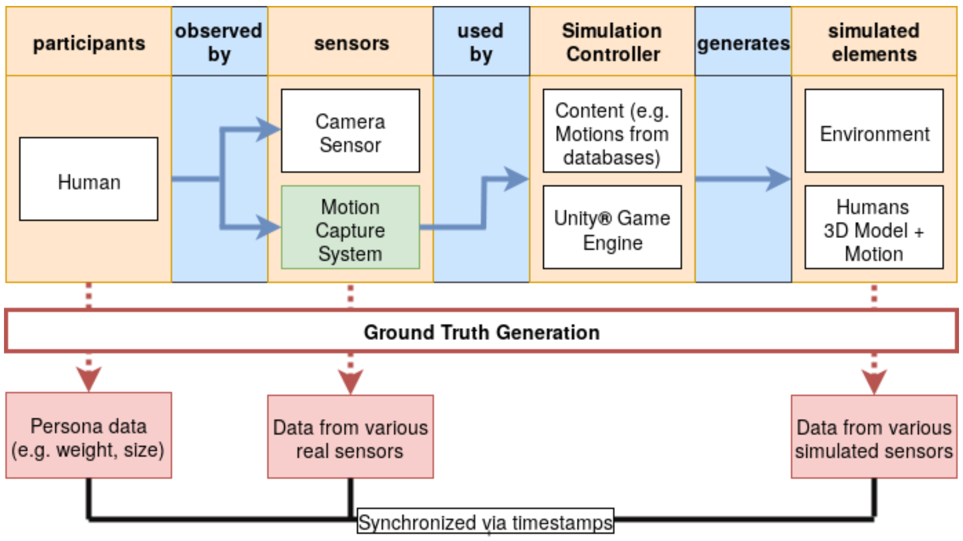


Figure 3.1: Architecture of the simulation-driven recording setup, showing the components to observe a human in a lab setup, obtain data from this human, and generate simulated sensor information enriched with this data. The kind of ground truth data produced by the different components of the simulation framework is displayed at the bottom.

2. The human models in the simulation can be perceived by various sensors, which can be placed freely around the humans. This allows fast adaption to various applications (for example, in-car sensors and infrastructure cameras).

3. In the simulation environment the positions, and properties of all elements in the three-dimensional space are known and can be used to generate annotation data automatically. For example, 3D human pose joints can be easily read out, and the animation files can be used to determine at any time which action a 3D avatar is currently performing, if annotated correctly.

4. It is possible to have camera sensors record how a person's movements are captured in a motion capture lab. Then, the lab environment, including the exact camera positions, can be simulated to evaluate how the simulation behaves compared to real sensor data under the same conditions.

3.3 Required Data To Generate Simulations

To generate appropriate simulations, the simulation environment requires content, such as 3D human models, animations, or environment models. The following shows how these required raw data for the simulation environment are obtained and generated.

3.4 3D Models

The appearance of simulation data is mainly influenced by the 3D models used. There are various possibilities to obtain and create corresponding models. For a basic simulation, the environment, 3D human models, and other 3D objects like vehicles are required. The following is a brief description of how the 3D model data for the simulation are obtained.

3.4.1 3D Environments

The 3D environment to be used depends strongly on the purpose of the simulation. For some experimental constellations, no actual environment is necessary. An example would be the work on corner cases (see section 4.1), in which a pose recognition algorithm should be taught rare human poses. It was important to vary the background strongly to focus on the human appearance and the skeleton configuration, so the background was generated using random 2D image data. A 3D environment was not used. When real 2D images are used as backgrounds, it is important to select them based on the training data requirements. For example, in human recognition algorithms, care must be taken that no humans are visible on the 2D background images since no annotations are generated for them. In other applications, the environment is very relevant. Such an application could be time-dependent algorithms like action detection, where scene consistency between frames is necessary. An example would be the detection of pedestrian actions like hitchhiking. It is important to have a scene with a street and a sidewalk, so that appropriate context information is available. There are numerous 3D environments that are available for free or can be purchased commercially. There is also the possibility to create the environments manually or to use 3D scanning. The 3D environments used in the experiments in sections 4.2, 4.3 and 4.4 are manually created or obtained from third-party providers.

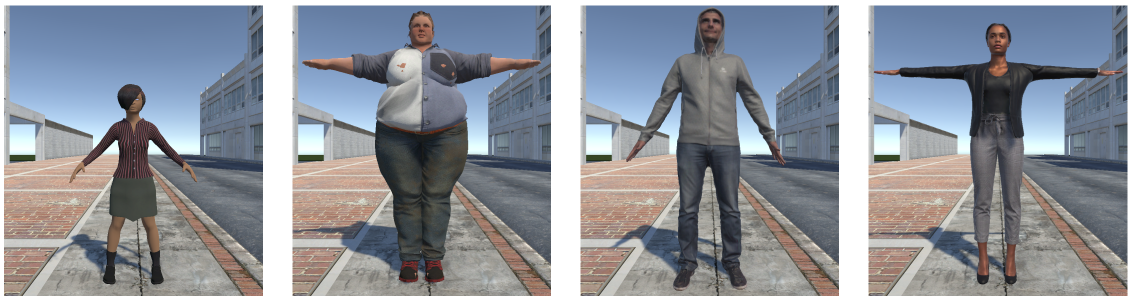
3.4.2 3D Objects

The use of 3D objects depends on the application requirements. For example, objects could be used to cover parts of bodies to make learning algorithms more robust against occlusions in real sensor data. In other applications, objects with which a human interacts could be relevant. An example could be a robot that is supposed to collaborate with a human and therefore is taught how a human interacts with different tools. Here it would be essential to use 3D models of the tools. Other examples would be crossing a street with a stroller or crossing a street with a trolley. The focus of this work is on the movement of people. Therefore, objects were not simulated, and machine learning algorithms should learn actions and intentions from the movement of the simulated people. Large databases with

3D objects of high quality already exist. For example, many vehicle models can be easily obtained. It is also possible to manually create the models or use 3D scanning methods for more specific requirements.

3.4.3 3D Humans

3D human models are the most important part of the simulation presented here, besides the motion data. Many visual appearances are required for the simulation of humans in the framework. The 3D human models are either created with a Vitronic VITUS^{bodyscan} full-body scanner[1], are generated by software such as MakeHuman[2] or the Reallusion Character Creator[3] at the sacrifice of realism or are obtained from a third-party provider (see Figure 3.2).



(a) Make Human[2] (b) Reallusion Character Creator[3] (c) 3D Scan (d) 3D scan with manual post-processing

Figure 3.2: Example 3D models from different sources and with different quality levels. (a) Procedurally generated with Make Human[2], enables fast generation but with low level of detail. (b) Procedural generation with the Reallusion character creator[3], higher level of detail but sometimes error fragments when using extreme values. (c) 3D scanned models from full-body scanner with automatic post-processing, rapid generation of models, but low fine details. (d) 3D scanned model with manual post-processing from a commercial supplier, high level of detail but costly in production.

The body scanner is laser-based, so unlike methods using photogrammetry, dark clothing does not pose any issues. With this system, a full-body scan can be performed in under ten seconds. To animate a 3D human model, it must first be rigged and skinned. Rigging means that an internal skeleton (rig) must be created

[1]https://www.vitronic.com (accessed on 2022-03-06)
[2]http://www.makehuman.org (accessed on 2022-03-06)
[3]https://www.reallusion.com/de/character-creator/ (accessed on 2022-03-06)

that represents the bones and joint structure of the model and determines the freedom of movement of the model accordingly. Once this skeleton is created, it is determined how the surface of the 3D model deforms based on skeletal movement, a process known as skinning. Autorigger [117] is used to automatically rig and skin the 3D scanned models. Besides the self-scanned models, there are several suppliers of scanned and rigged 3D models. We use several 3D models acquired in this way, as they are of much better quality than software-generated models, and the respective provider is in charge to fulfill the GDPR data protection requirements. With self-scanned models, the legal situation is often not clear. This problem is mostly present in very high-resolution models where biometric details, such as the iris, are mapped in detail (Article 9 EU GDPR). In addition, with self-scanned models, scanned persons can withdraw their consent to the publication of the data (Article 7, paragraph 3 EU GDPR), which makes it more difficult to provide data in a dataset or benchmark. In addition, metadata is also collected. This includes the age, weight, height, gender, and skin color of the 3D human avatar. These are used as filter options in the simulation and can be output as ground truth labels for sensor data.

3.5 Animations

Human avatars need to be animated to perform actions in a simulated environment. Literature divides animations into three categories [118, p. 64]:

Keyframe animation originated from the early approach to animation, where each frame was drawn manually. Frames can be interpolated between two keyframes in keyframe animation to reduce the manual effort. This type of animation allows a high degree of control over the animation but requires expert knowledge to make movements look realistic [118, p. 64, 119, p. 2]. Besides the creation of the keyframes, the type of interpolation is important. A linear interpolation usually leads to unnatural-looking movements because human movements are noisy and follow a slightly circular path [120, p. 62]. It is difficult to get variations in the animations due to the static nature of keyframes.

Procedural animation is a category under which various methods for automatically creating animations are summarized. Predefined procedures generate the animations. These procedures can have various rules, constraints, and parameters [121, pp. 141-142]. Examples are animations generated by applying inverse kinematics based on the current environment or physics-based simulations. There are also data-driven character animation techniques[4]. For

[4]It must be noted that DNN based animation techniques are categorized as procedural animation, while usually requiring large amounts of motion capture data to be trained on.

example, machine learning techniques can be used to learn various motions as well as transitions between motions based on data. User input, such as a direction or a velocity with various constraints from the environment and the skeletal structure, can be used as input for such neural networks to synthesize realistic movements [122, 123, 124]. Procedural methods enable physically realistic movements and allow their adaptation, for example, a change in walking speed [125, p. 72]. Furthermore, it is possible to react interactively to environmental details such as obstacles or other people. Since parameters, rules, and constraints of movements are given, it is also possible to include secondary movements (for example, of hair or clothing) using these parameters [125, p. 72]. However, the same rules, constraints, and parameters always result in the same movements.

Motion capture animation describes animations that are created by tracking and recording movements. There are various technologies to achieve such tracking. Examples are IMUs or optical tracking systems. The process of motion capturing usually consists of capturing motions, independent of the appearance [119, p. 51]. The motion is then transferred to digital models. This process is called retargeting. Motion capturing usually allows the fast creation of animations but often requires expensive hardware and advanced post-processing technologies such as retargeting systems. High-quality motion capture systems capture fine details of movements. Thus, it creates many variations when recording the same movement multiple times with multiple people. Since the animation is fully based on recorded data, it is not possible to define important properties of the motions and often hard to determine how to edit the motion data to achieve a desired effect [119, p. 53]. Editing motion capture data is sometimes required just because of errors in the recording process. The editing itself is usually complicated because data in every time frame has to be edited. The challenges associated with editing motion capture data also mean that it is inflexible in use and difficult to adapt to new environments or situations.

It is possible to combine approaches of these three categories. For example, motion capture data can be adapted via procedural methods to react dynamically to environmental details. An example would be to decouple the neck joint from the animation and let the head follow an object in the virtual scene. Motion capture data can also be adjusted using inverse kinematics to simulate a step from the sidewalk to the street, even though the data was recorded on a flat surface.

Since the motion data is to be used in a simulation to train data-driven algorithms, one main requirement is the variability in the execution of movements. If certain movements are always executed in the same way, overfitting problems are likely to occur. For example, a neural network may learn the exact execution of an action, leading to recognition problems in case of deviations of this execution. By capturing motion several times and recording several people, one automatically gets a high variance of the same movement. If only the type of movement is specified, but the execution of the movement is not restricted, even executions can be recorded that individual persons might not even think of. Movements that are recorded by motion capturing can also be transmitted live to a simulation. This allows online validation of algorithms, which could be used to determine, for example, whether an algorithm has problems with certain types of actions or a certain type of execution of an action. The use of motion capturing presents a corresponding challenge in the design of scenes. Due to the static nature of the recordings, it is not a simple task to transfer movements to other scenarios or represent them realistically in different environments. By using approaches like inverse kinematics, some of these problems can be solved, and some of them are irrelevant due to the fast producibility of motion-capture animations. Despite these disadvantages, motion capturing is the best method for creating animations in the use cases, due to the realistic animations and high variability in the movement execution by several recorded people.

Nevertheless, large-scale motion capture data acquisition is a complex process. Recordings have to be made by several people who have to perform complex motion sequences with individual characteristics. In addition to the actual recording, the recording must also be prepared by creating a recording catalog for each person to be captured, for example, by providing data protection agreements, experimental consent forms, and selecting concrete sequences to be captured. Each individual recording of a movement sequence places high cognitive demands on the supervisor and the person being recorded. Information must be perceived, such as the movement to be performed, the speed of movement execution, the start position, which accessories are to be used on which body part, and which specific steps are to be executed. In addition, appropriate post-processing steps must be taken after each recording. Thus, available information about the content of a recording must be assigned to the actual motion data. Figure 3.3 highlights the recording process for a single participant.

The most time-consuming steps in this process are as follows:

- Preparation: The creation of a recording catalog is done by randomly compiling a catalog of n concrete scenes from a large general catalog of possible sequences, accessory combinations, movement speeds, etc. The catalog is then used as a basis for the creation of the recording catalog. The individual elements in this catalog must be assigned unique identifiers (UIDs) that are also used as identifiers for the motion capture files and thus enable the matching of metadata and recorded movements.

- Record: During recording, the participant must be told for each sequence what is to be recorded, in what form, and with which boundary conditions.

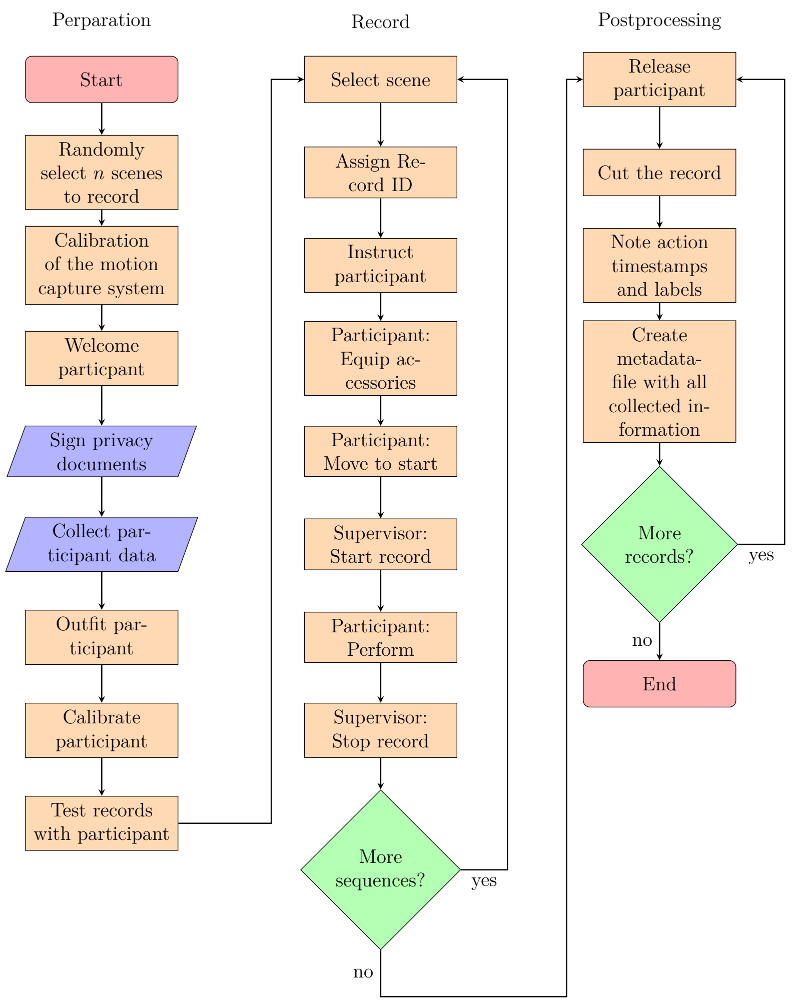


Figure 3.3: Procedure for a motion capture session with one participant, including preparation and post-processing.

The participant must then equip appropriate accessories, go to the starting position, mentally internalize the steps to be performed and carry out the recording.

- Post-processing: In post-processing, each recording must be qualitatively judged retrospectively to cut superfluous parts, to individual label actions in the overall sequence, and, finally, to generate a corresponding metadata file for each recording, in which it is indicated, for example, which sequence was recorded, which individual actions are contained, which characteristics the participant had. The metadata file must be created in such a way that it can be used as a basis for the recording.

Going further, some parts of this process are particularly prone to error. Due to numerous short sequence recordings, it is often difficult for participants and supervisors to maintain concentration over an extended period. Therefore, during test recordings, sources of errors frequently occurred in the following areas:

- Preparation errors: When manually creating recording catalogs, the supervisor selects randomly from the overall catalog, and individual scenes from the comprehensive catalog may be forgotten.
- Record errors 1: During recording, the supervisor may slip in the line for individual components of a recording description or entire record descriptions. As a result, something is recorded that no longer matches the description in the recording catalog.
- Record errors 2: It often happened that side mix-ups, such as cell phone in the right hand instead of the left hand.
- Record errors 3: In some cases, the UIDs from the recording catalog were not entered correctly, which meant that recordings could no longer be assigned to the corresponding metadata.
- Post-processing errors one: When labeling individual actions in a sequence, there were sometimes typing errors in the corresponding time specifications.
- Post-processing errors two: When creating the metadata file for each recording, typing errors can occur in the UIDs or actual metadata, recording sequences can be mixed up by slipping in the list, or individual information can be forgotten to be included in the metadata file.

To minimize the manual time required as well as the possible sources of error, a motion capture management system was generated as part of this work. It supports the generation and management of recording catalogs, guides the actual capture, provides information to the participant and supervisor, automatically links recorded data and metadata, bundles the information, and makes it available for use by the simulation framework. Further, to support post-processing, an action labeling tool was created and integrated directly into Vicon's motion capture software.

3.5.1 Motion Capture Manager

The motion capture management system is designed to support the user in all three phases of a motion capture session. The system is optimized for the situation in the motion capture lab of Reutlingen University to record different scenes of pedestrian movements. Figure 3.4 shows the motion capture lab from the perspective of a participant. There are two large monitors visible to both the participant and the supervisor. The supervisor's workstation is at the end of a row of workstations,

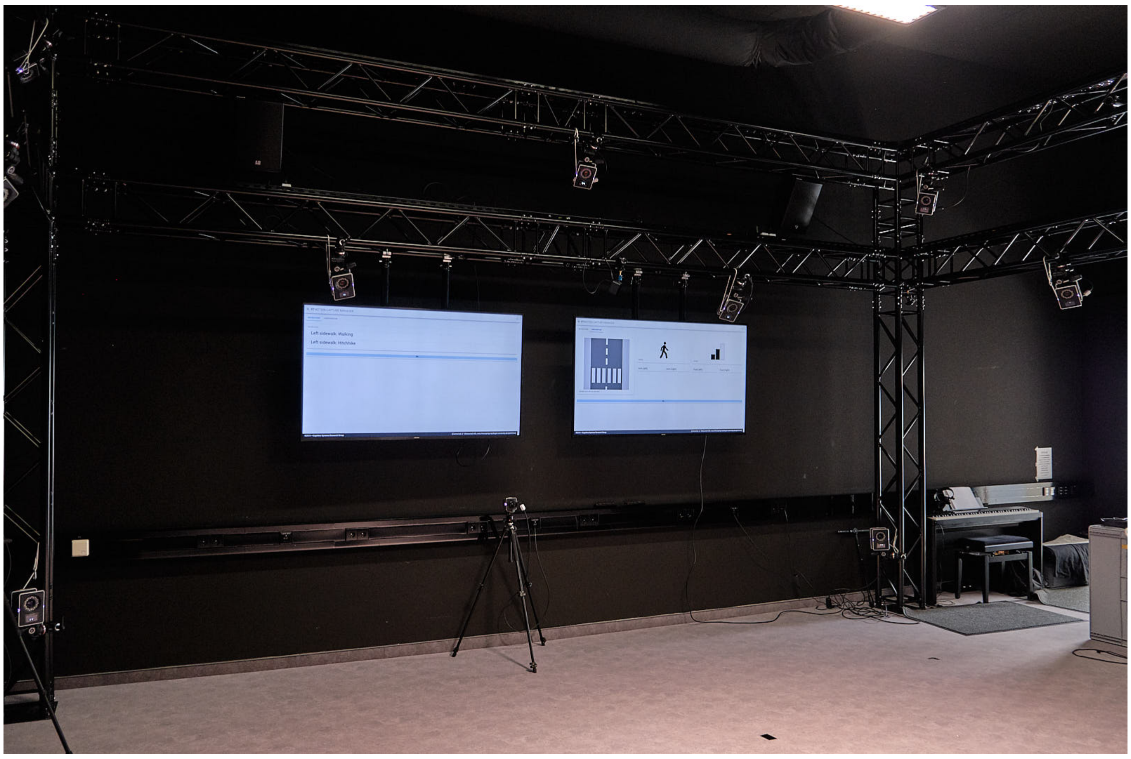


Figure 3.4: Reutlingen University Motion Capture Lab: Participants' View

so it should be avoided that the supervisor has to get up and go into the motion capture volume to instruct the participant. In the picture, the motion capture management system can already be seen supporting a record by displaying all relevant information on the two screens.

The requirements for motion capture recordings were designed primarily for pedestrian movement recording. However, the system can easily be extended for other use cases by adding further master data, new master data types, or simply creating a new recording catalog. In the following, it will be shown how exactly the motion capture management system supports the three phases of a motion capture recording session.

3.5.1.1 Preparation

The focus of the preparation phase is to create a recording catalog for the participant. For this purpose, the motion capture management system supports the following master data:

Genders Possible genders for participants; currently, only females and males are used.

Skin colors Skin colors are used to describe the skin color of a participant. This information is mostly useful if a 3D full-body scan is created from this participant in addition to motion capture records. Within the scope of this work, the Fitzpatrick skin phototype[5] is used as a categorization of skin colors.

Accessories Accessories that participants can use. Examples are a smartphone, an umbrella, or a stroller.

Human body parts Human body parts are used in combination with accessories. Examples would be the right and left hand or the right leg.

Impairments Possible impairments of human body parts. Examples would be fractures or sprains.

Human configurations Human configurations are used to manually control combinations of accessories, injuries, and human body parts. They describe an actual human state. One example would be a person with an injury on the right leg and a crutch in the left and right hand.

Movements Movements are used to define a general movement in a scene. Examples would be standing, walking, or running.

Movement speeds Movement speeds are a further subdivision of movements. They define, for example, that a movement should be executed slowly or quickly. Only adjectives such as fast or slow are deliberately used so that the participants execute the movement as they interpret the corresponding adjective.

Scenes Scenes define the basic environment in which the movement is to be executed. The participant should build up a mental image of the environment through the scene description and move accordingly. An example would be a straight street with two sidewalks.

Locations Locations are used to control the positioning of a participant. They are used in combination with scenes. For example, if a scene contains two sidewalks, two possible locations would be the left or right sidewalk.

This master data must be created in the system once and is used when creating recording catalogs. Pictorial descriptions are used in addition to textual descriptions for movements, movement speeds, accessories, and scenes. The pictograms are displayed via a user interface during recording (see Figure 3.5). To create a recording catalog, the sequences to be recorded must first be described. A sequence always describes motion sequences for a specific scene. It consists of movements, movement speeds, human configurations, and a list of instructions, each consisting

[5] `https://dermnetnz.org/topics/skin-phototype/` (accessed on 2022-03-06)

of a location and a description. It should be noted that several movements, movement speeds, and human configurations can be used for a sequence. The system can randomly mix these to achieve a greater variation. Accordingly, a recording catalog is composed of n sequences and differs from project to project. The user can choose a certain amount from the list of sequences and create a project description from it, supported by the system, that specifies how many concrete recordings are needed to record each sequence at least once. This information can then be used to determine how many participants are needed to complete the related project. All described steps must be executed once at the beginning of a project. During the preparation of a recording, the supervisor only needs to create a participant and create a record plan by selecting the participant and the associated project. The system will then automatically select a predefined number of recordings randomly from the project's entire recording catalog and create corresponding entries for the recordings in the database. Once the recording plan has been created for the specific participant, the system can also control the specific recording sessions.

3.5.1.2 Recording

By creating a recording catalog for the participant, the system knows all the variables for each sequence to be recorded. This information can be displayed directly by the system during the recording sessions. This eliminates the need for the supervisor to instruct the participant before each recording exactly what to do in the next sequence. The two screens in the motion capture lab are used for this purpose (see Figure 3.4). On one of the screens, the individual steps to be performed are displayed. An example would be the three steps of 'walking straight on the sidewalk', 'looking for traffic', and then 'crossing the street'. The second screen displays the boundary conditions for the shot, for example, the scene, the movement, the movement speed, and the accessories to be used. An example of this screen is shown in Figure 3.5.

In addition to this information, the supervisor is shown a recording ID with a copy option on a separate screen. This ID must be taken over in the motion capture software and set for the current recording. In this way, all files belonging to this recording are provided with the corresponding ID and can thus be assigned to the database entry with the sequence description. The ID is not the internal UID from the database, but a continuous integer ID because the standard motion capture programs offer a possibility to iterate integer values in recording names automatically, so the ID must be set only once at the beginning and on repetitions.

By using this symbolic view, all boundary conditions for a record sequence can be quickly captured. The participant looks at the information, gets the required accessories, goes to the start position as indicated in the instructions, and is then ready to perform the record. The supervisor can remain at the workstation and monitor the recording process, but does not have to give any further instructions.

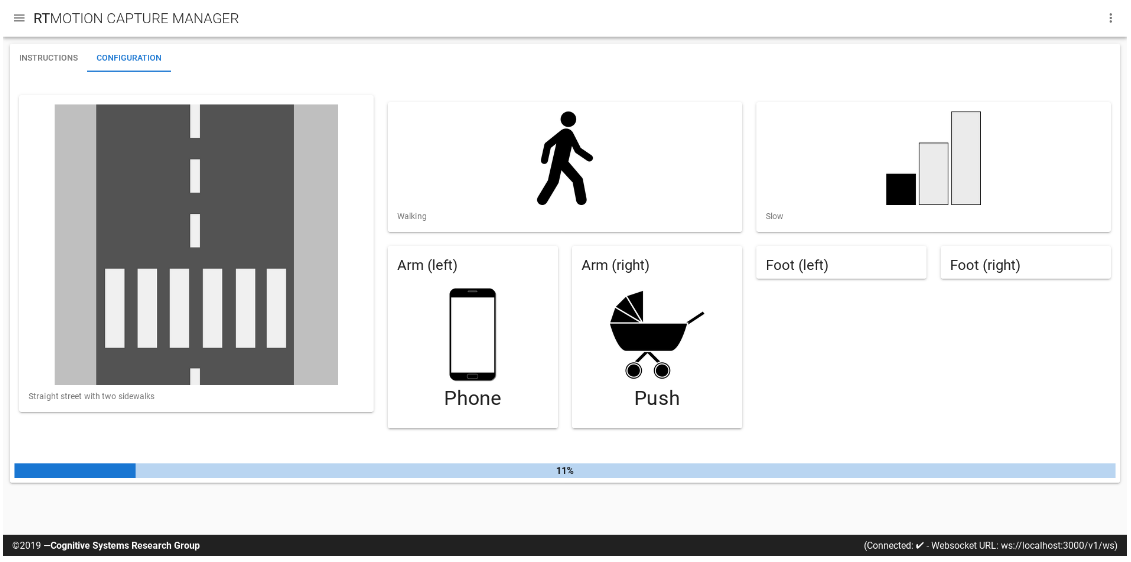

RTMOTION CAPTURE MANAGER
INSTRUCTIONS
CONFIGURATION
Straight street with two sidewalks
Walking
Slow
Arm (left)
Phone
Arm (right)
Push
Foot (left)
Foot (right)
11%
©2019 –Cognitive Systems Research Group
(Connected: ✔ - Websocket URL: ws://localhost:3000/v1/ws)

Figure 3.5: Motion Capture Manager: Sequence conditions during a record. The blue bar shows the total progress of the record session.

He only controls the start and stop of a recording, or initiates a repetition if necessary.

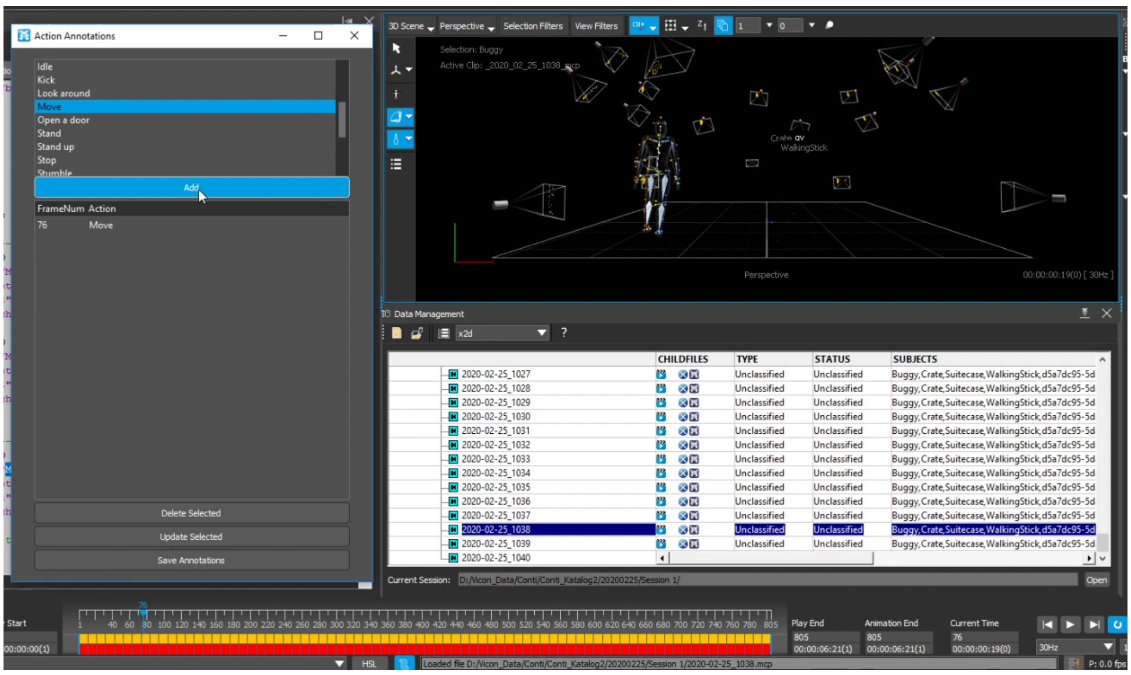
We have used the motion capture manager to record a large-scale pedestrian action recognition dataset. Details about the required recording times are presented in section 4.4.4.3.

3.5.1.3 Post Processing

The time-consuming creation of metadata for the recordings in post-processing becomes obsolete through the use of the motion capture management system. All metadata is available to the system and can be exported directly accordingly and assigned to the corresponding recording data utilizing the UID. The only time-consuming step that cannot yet be automated with the motion capture management system is the generation of action labels within a sequence. However, a labeling tool has been created for this purpose, which is intended to simplify labeling. The tool was integrated into the interface of the Vicon Shogun[6] software, in which the user can view the motion capture data and scroll through the time sequence as desired (see Figure 3.6).

The labeling tool provides a list of all individual actions, such as walking, looking around, or stumbling. The user can then go to a specific point in the recording, select an action and add it to that point in time. It is also possible to select multiple individual actions simultaneously, for example, running and looking around. The

[6]`https://www.vicon.com/software/shogun/` (accessed on 2022-03-06)



Idle
Kick
Look around
Move
Open a door
Stand
Stand up
Stop
Stumble

FrameNum	Action
76	Move

	CHILDFILES	TYPE	STATUS	SUBJECTS
2020-02-25_1027		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1028		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1029		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1030		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1031		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1032		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1033		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1034		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1035		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1036		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1037		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1038		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1039		Unclassified	Unclassified	Buggy,Crate,Suitecase,WalkingStick,d5a7dc95-5d
2020-02-25_1040				

Current Session: D:/Vicon_Data/Conti/Conti_Katalog2/20200225/Session 1/

Start 00:00:00(1)
Play End 805 00:00:06:21(1)
Animation End 805 00:00:06:21(1)
Current Time 76 00:00:00:19(0)

Loaded file D:/Vicon_Data/Conti/Conti_Katalog2/20200225/Session 1/2020-02-25_1038.mcp

Figure 3.6: Action labeling tool, which is directly integrated in the motion capture software of the Reutlingen University Motion Capture Lab.

selected actions are then valid until other actions are selected later or the recording stops. Besides the concrete actions, there is a particular keyword ‘Stop’, which is used as an indicator for the stop of the recording sequence. This is necessary if the recording was stopped after the sequence was completed and contained movements that did not belong. The tool then exports the information in which time individual actions took place into a metadata file, which can be assigned to the actual recording and the other metadata through the file name. In the end, the metadata from the labeling tool and the motion capture management system are merged via a script, packed together with the exported animation, and can then be used directly in the simulation environment.

3.6 Asset Database

The asset database represents the database of the simulation. All required raw data and the metadata must be processed and stored in this database. In the Unity game engine, the asset database is file-based. In this work, the mapping between raw data and metadata is also file-based by using the same file name for raw data and metadata. The file extension defines the type of data. Table 3.1 summarizes which data was used in the simulation during experiments in this work.

The metadata files are in standard JSON format, which is supported by all

Data	**3D human models**
Metadata	Age, weight, size, gender, skin color, UID
Sources	3D scanned, MakeHuman, 3rd party
Data	**Human animations**
Metadata	Age, weight, size, gender, skin color, UID, accessory usage, instructions, movement, movement speed, actions per frame, project, scene, sequence description
Sources	Motion capture system, CMU motion database
Data	**3D Objects**
Metadata	Dimensions, color(s)
Sources	3rd party
Data	**Environment**
Metadata	Classes, object identifiers
Sources	Images, hand-modeled, third party

Table 3.1: Overview of all data in the asset database. For each data object, there can be corresponding metadata that contains further information about the object. The sources of the data in the simulation used are also indicated. The exact data used, and its sources are specified in the respective experiment descriptions.

major programming languages. Thus, the metadata files are easily usable in scripts and applications as well. An example for a motion description file is provided in Appendix B.2 and an example for the character description file is provided in Appendix B.1.

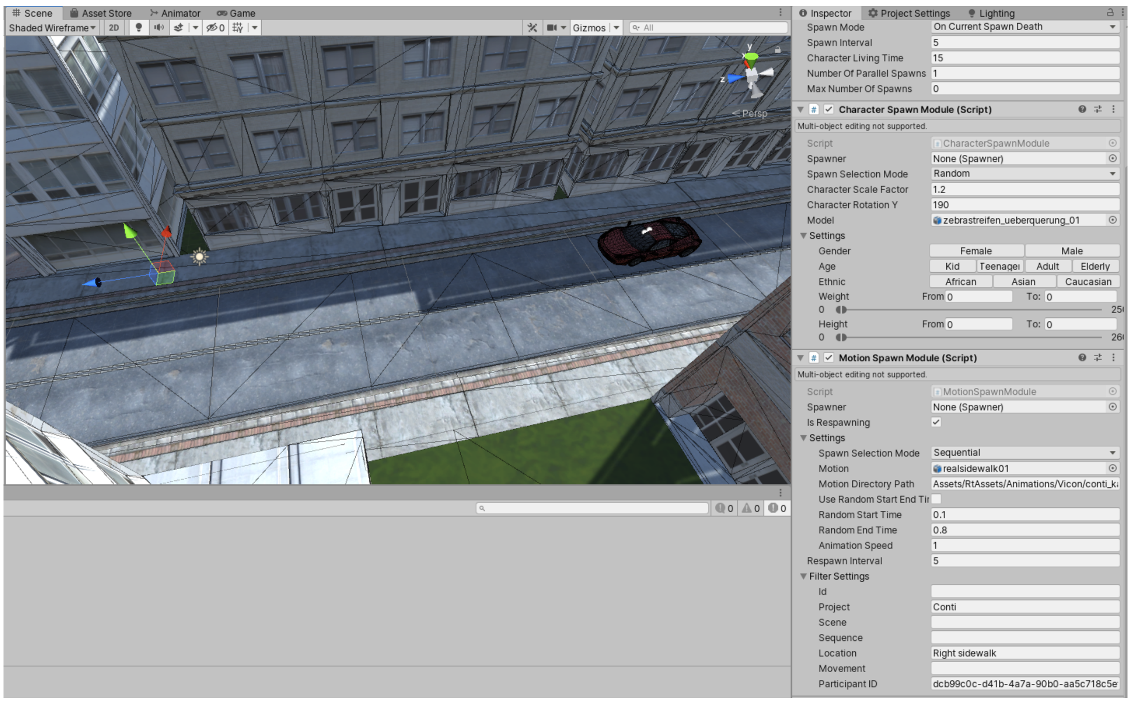
3.7 Simulation

The simulation is based on the Unity game engine. The simulation's primary purpose is that it offers to generate highly generic scenes easily with various environments, human models, and motions. In addition to these main components, various other parameters, like the position, light intensity, and cameras, can be varied, resulting in different simulations. The import consists of the actual data, 3D models, motion capture files, and their corresponding metadata, which will be appended to the actual data in the game engine. So, for example, a 3D model would have properties like age, weight, height, and motions could have properties like intention or executed actions. The metadata of 3D models and motions are integrated directly into the Unity user interface as filter options (see Figure 3.7). This allows the user to select, for example, a specific gender for the 3D model and an age range for the 3D model and motion. In addition, scene-specific parameters such as the time of day, lighting, or the possible position of 3D models can be defined.

3.7.1 Asset Import

During the import of 3D models and animations, various settings are applied to use the elements in the simulation without further manual intervention. For example, layer masks are automatically set for 3D models and a categorization flag for the output of segmentation data. For human models, the following further adjustments take place:

- Automatic creation of an animator, which controls the character's animation.
- Automatic setting of basic animation settings. This includes the application of root motion and the addition of a grounder, which uses inverse kinematics to ensure that the character's feet remain on the ground. This is important, for example, when stepping off a curb.
- Setting a cylindrical collider based on the size of the 3D model read from the metadata.
- Setting the physical model based on the weight of the 3D model read from the metadata.



```
Spawn Mode                    On Current Spawn Death
Spawn Interval                5
Character Living Time         15
Number Of Parallel Spawns     1
Max Number Of Spawns          0
Character Spawn Module (Script)
Multi-object editing not supported.
Script                        CharacterSpawnModule
Spawner                       None (Spawner)
Spawn Selection Mode          Random
Character Scale Factor        1.2
Character Rotation Y          190
Model                         zebrastreifen_ueberquerung_01
Settings
  Gender                      Female | Male
  Age                         Kid | Teenage | Adult | Elderly
  Ethnic                      African | Asian | Caucasian
  Weight              From 0          To: 0
  0                                              25
  Height              From 0          To: 0
  0                                              26
Motion Spawn Module (Script)
Multi-object editing not supported.
Script                        MotionSpawnModule
Spawner                       None (Spawner)
Is Respawning                 ✓
Settings
  Spawn Selection Mode        Sequential
  Motion                      realsidewalk01
  Motion Directory Path       Assets/RtAssets/Animations/Vicon/conti_k
  Use Random Start End Ti     
  Random Start Time           0.1
  Random End Time             0.8
  Animation Speed             1
Respawn Interval              5
Filter Settings
  Id
  Project                     Conti
  Scene
  Sequence
  Location                    Right sidewalk
  Movement
  Participant ID              dcb99c0c-d41b-4a7a-90b0-aa5c718c5e
```

Figure 3.7: 3D Human avatar and motion filter options included in the Unity user interface. The filtering is based on the collected metadata.

- Adding scripts that provide ground truth information from the character data (see section 3.7.4).
- Assignment of skeletal elements to the ground truth skeletal model (see Figure 3.13), if necessary, with the calculation of human joints not included in the rig.

When importing animation data, settings are automatically made that tell the Unity game engine that it is a humanoid animation. This allows it to be used in the internal Unity retargeting system to animate 3D human models. This internal Unity retargeting system usually works very well and allows one to use arbitrary animations with arbitrary 3D human avatars. However, care must be taken that the body dimensions are not significantly different from the person from which the animation data originates to the 3D human model. Otherwise, errors such as limbs passing through the torso may occur (see Figure 3.8).

The metadata is also used to read out which actions the animation file contains. Based on this data, the animation file is automatically trimmed to remove unwanted movements. For example, an animation could include a T-pose at the beginning or movements at the end that do not belong to the actual recording.

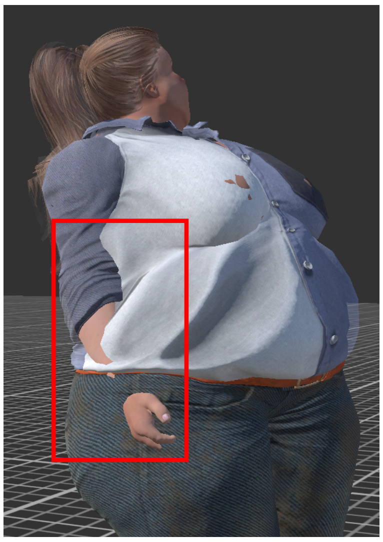


Figure 3.8: Example of a retargeting error. The movement was recorded by a slim person and transferred to the avatar of a more corpulent person. Since the arm of the slim person was close to the body and the retargeting process does not consider the shape of the target model, the arm now goes through the torso.

3.7.2 Content Generation

Content spawn modules mainly control the generation of a scene. As a base, scenes that contain only the environment model and static objects are used. All other elements, such as pedestrians, are created by content spawn modules during the execution of the simulation. Each content spawn module is responsible for generating a specific object type in the scene. A content spawn controller is responsible for the timing and execution of the actual content spawn modules. Several content spawn modules can exist in one controller. This allows the controller to create objects that are time-dependent on other objects. An example of this is a particular lighting situation that should be constant while creating a predefined number of human models. Most of the modules provide the possibility to create their content randomly, sequentially, or by explicitly selecting the object to be used. Currently, the following spawn modules have been implemented:

Character Spawn Module A character spawner handles the creation of 3D character models. The models can be from different sources (see section 3.4.3). Metadata descriptions of each model are required to enable model filtering and the automatic provision of annotation data. 3D models can be selected directly or randomly with possible restrictions by filtering descriptions.

Motion Spawn Module Motion spawn modules look for available 3D models spawned in the same spawner context and apply a specific motion as animation on them. Animations in this work are retrieved from the CMU Motion Database[7] or are self recorded (see Section 3.5).

Position Spawn Module Objects created by a spawner can be positioned in the environment by using the position spawn module. It is possible to use some predefined positions. It is also possible to use random placement, which requires defining some areas where objects are allowed to be placed, for example, by using a tag on the 3D environment models like 'left sidewalk'. Additionally, it is possible to define a minimum required space to place an object. This can be useful to give a character enough room to play its animation without colliding with walls or other objects.

Skybox Spawn Module *'Skyboxes are a wrapper around your entire scene that shows what the world looks like beyond your geometry.'*[8] So basically, in a simulation, the main contribution of a skybox is the look of the sky and the delivery of environmental lightning. Skyboxes can be freely exchanged to simulate different environmental conditions like weather or time of the day.

Light Intensity Spawn Module Usually, there are predefined light sources, like the sun or lamps, in a scene in addition to the environmental lighting. The light intensity spawn module can control these light sources. It can set the intensity of these lights randomly, progressively, or based on the current skybox.

3.7.2.1 Spawner

A so-called spawner controls the spawn modules. A Spawner contains several spawn modules and handles the creation, lifetime, and destruction of elements in the simulation. The timing of spawns can be set by either a fixed interval or the lifetime of spawn objects. If all objects of the current spawned iteration are 'dead', a new generation will be spawned. The tasks of a spawner can be summarized as follows:

1. Spawn modules register themselves on a specific spawner. Modules that should share timing must use the same spawner. Modules can assign themselves priorities to handle the spawn order. This is, for example, required by motions and animations, which require a spawned 3D model to be applied.
2. For each frame, the spawner checks if some spawned objects should be deleted based on their lifetime. If so, the objects are destroyed and removed from the spawn objects.

[7]`http://mocap.cs.cmu.edu/` (accessed on 2022-03-06)
[8]`https://docs.unity3d.com/Manual/class-Skybox.html` (accessed on 2022-03-06)

3. If the current time is higher than the next spawn time, new objects will be spawned. The number of objects that can be spawned simultaneously is only limited by hardware. For each parallel spawn, a container is created, which will hold all objects from the spawn modules. For each container, each spawn module will spawn its objects in the container.

4. Each spawn module can assign a lifetime to its spawned object(s). It sends this time to the spawner, which will destroy the objects after their time of death. The lifetime can also be used for local timing, for example, to trigger a new spawn if all objects of the current spawned iteration are dead. If a module does not deliver a lifetime, the objects will be alive until another spawn is triggered.

5. The spawner assigns an UID to each spawned object, which is used to identify objects in the outputted ground truth data.

3.7.3 Virtual Sensors

The simulation environment currently supports camera sensors. The Unity game engine already offers a physical camera model, which can be used for standard focal lengths. The focal length and the sensor size can be adjusted, determining the field of view. Since fisheye cameras are often used in the automotive environment, the simulation framework supports them through cubemaps generated from five different virtual camera sensors. This ensures a 360° panoramic view, and fisheye cameras with different field of views can be simulated based on this 360° view.

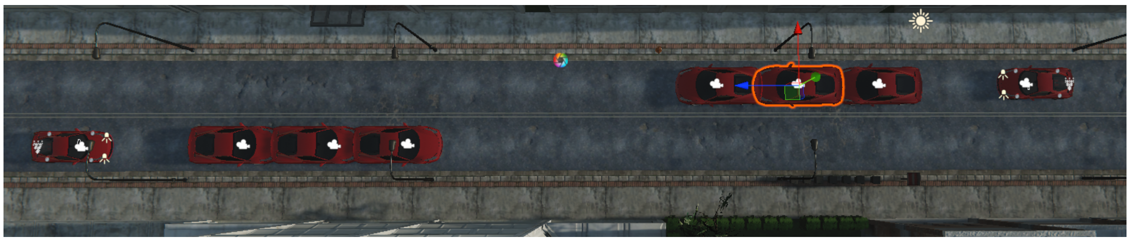


Figure 3.9: Top-down view of a simulated scene which contains multiple cars with front facing camera sensors. The cars would normally occlude each other, but the vehicle cameras mask out all other cars, which allows recording a scene from eight different cars in parallel.

The Unity game engine offers a layering system, whereby elements in the simulation can be grouped, and different actions and reactions can be included based on the layers. This is used in the virtual sensor environment to mask unwanted objects from a virtual camera image. For example, various vehicles can be inserted into a simulation that would typically occlude each other (see Figure 3.9). The other unwanted vehicles in the vehicle cameras can be masked using an appropriate culling mask. This makes it possible to record parallel data for different vehicles in different positions without other vehicles being included in the respective sensor data.

3.7.3.1 Camera Sphere

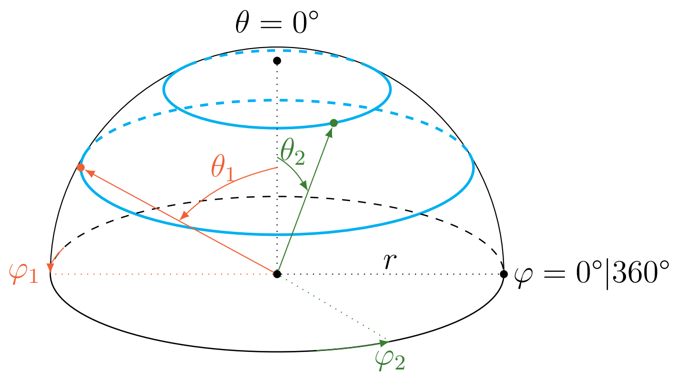


Figure 3.10: Example of a camera sphere with $r = 3$, $\theta_1 = 60°$, $\theta_2 = 30°$, $\varphi_1 = 180°$ and $\varphi_2 = 300°$.

One main benefit of a simulation environment is the possibility of freely placing a sensor in the scene. Some use-cases require specific sensor locations, for example, in a car's front window. Other use-cases just require many perspectives to reach a good generalization. One example of such a use-case is the corner-cases experiment (see section 4.1), in which many sensors used to capture the dataset are handheld, for example, smartphone cameras. The simulation framework supports such use-cases by providing a camera sphere generator. This generator provides a method to place camera sensors in spheres around a given target (see Figure 3.10). Using spheres provides the possibility to control the camera placement just by the three parameters radial distance r, polar angle θ and the azimuthal angle φ (see equations 3.1-3.3). The following definitions follow the Unity game engine's convention of a y-up, left-handed coordinate system.

$$\mathbf{p_x} = r\cos(\varphi)\sin(\theta) \tag{3.1}$$
$$\mathbf{p_y} = r\cos(\theta) \tag{3.2}$$
$$\mathbf{p_z} = r\sin(\varphi)\sin(\theta) \tag{3.3}$$

After positioning the camera, it needs to be rotated towards the object, in the use case, the human. The Unity game engine uses a left-handed y-up coordinate system as well as column-major matrix notation. Let $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ the camera coordinate system axis representing the right, up and forward vectors, **z** is calculated from camera position **p** and target position **t** (see equation 3.4). The up vector **u** (in the Unity game engine $[0,1,0]^T$) is required to calculate the last two axes of the camera coordinate system (see equations 3.5 and 3.6).

$$\widehat{\mathbf{z}} = \frac{\mathbf{t}-\mathbf{p}}{|\mathbf{t}-\mathbf{p}|} \tag{3.4}$$

$$\widehat{\mathbf{x}} = \frac{\widehat{\mathbf{z}} \times \mathbf{u}}{|\widehat{\mathbf{z}} \times \mathbf{u}|} \tag{3.5}$$

$$\widehat{\mathbf{y}} = \widehat{\mathbf{z}} \times \widehat{\mathbf{x}} \tag{3.6}$$

The axes of the camera's coordinate system are the parameters for the rotation part of the so-called LookAt matrix, which is composed of the rotation matrix and the translation matrix based on the position vector of the camera [126, p. 96].

With the defined camera coordinate system, it is straightforward to rotate the camera object accordingly using the LookAt rotation matrix R shown in equation 3.7.

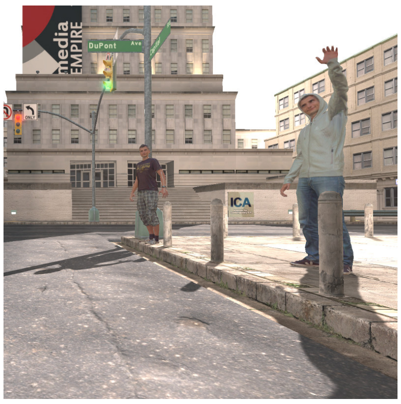
$$R = \begin{bmatrix} \widehat{\mathbf{x}_\mathrm{x}} & \widehat{\mathbf{x}_\mathrm{y}} & \widehat{\mathbf{x}_\mathrm{z}} & 0 \\ \widehat{\mathbf{y}_\mathrm{x}} & \widehat{\mathbf{y}_\mathrm{y}} & \widehat{\mathbf{y}_\mathrm{z}} & 0 \\ \widehat{\mathbf{z}_\mathrm{x}} & \widehat{\mathbf{z}_\mathrm{y}} & \widehat{\mathbf{z}_\mathrm{z}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

The simulation frameworks support a random assignment of the parameters r, θ, and φ as well as just a definition of how many camera sensors should be placed in an even manner.

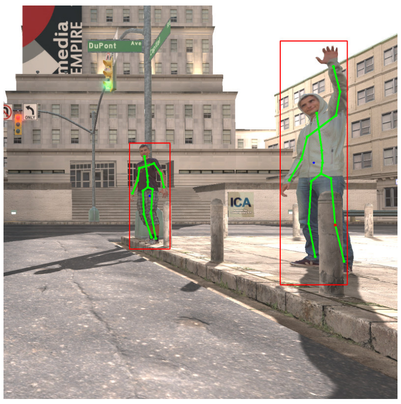
3.7.4 Ground Truth Data Generation

Immediately available ground truth data is an essential part of the simulation and is the direct interface with which to train and validate the perception module of an autonomous system. In principle, each element is known within the simulation and can be annotated as required. Examples of generated ground truth data are visualized in Figure 3.11.

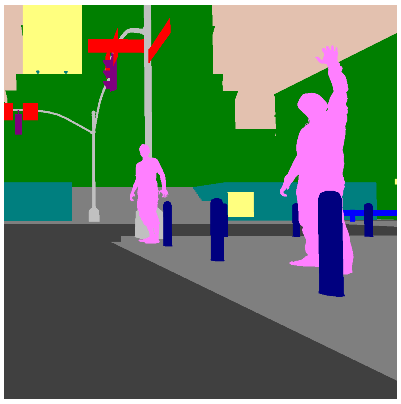
The corresponding raw camera image is shown in Figure 3.11a. In general, every object in a simulated scene is known in detail, and each object can be uniquely identified at any time. This is essential for object-level segmentation, action detection, and tracking, where a lot of detailed information about an instance of an object and its current state is required. In terms of human models, every bone in the underlying skeleton is known. This information can be directly used to annotated human joints and may indirectly be used, for example, to calculate bounding



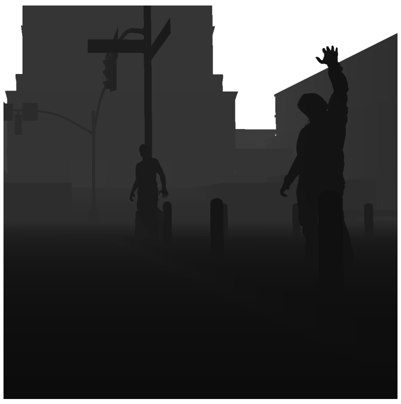
(a) Simulated Raw Sensor Data



(b) Bounding Box & Human Pose



(c) Semantic Segmentation



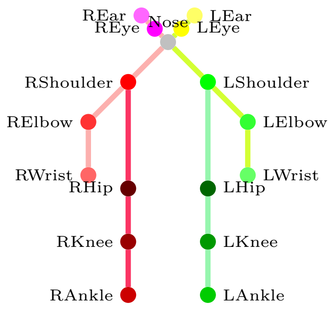
(d) Simulated Depth Data

Figure 3.11: Example of a simulated scene with visualized ground truth.

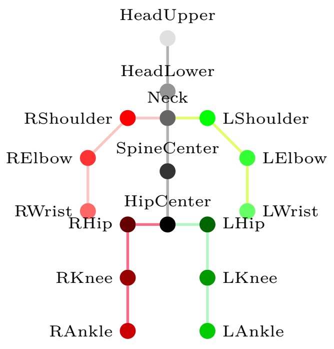
boxes. All of this information enables us to generate sensor data which have annotated classes for semantic classification, 2D and 3D bounding boxes for detection (Figure 3.11b), 2D and 3D human joints for human pose estimation (Figure 3.11b), pixel-wise classes for semantic segmentation (Figure 3.11c) as well as depth information for every pixel (Figure 3.11d). Furthermore, human metadata is added in each frame for each human instance, which may be represented, for example, by a bounding box or human pose. This leads to a list of humans and their corresponding metadata generated for each camera image. One example for this kind of ground truth is action and intention information added to a human instance, for example, a frame where a human performs the action 'walking' with the intention 'crossing the street'. The creation of ground truth data in the simulation involves collecting the information and outputting this information in a suitable form. Some ground truth data is internally based on the same collected data. Examples are the 2D and 3D human poses and the bounding boxes of humans, which are also based on this pose data. In the following, it will be shown which ground truth types are supported by the simulation and how they are obtained.

3.7.4.1 3D Joint Positions and Orientations

It is important to define a skeletal structure for pose ground truth data. As described in section 1.3, there are various datasets which differ in the used skeleton structure (*cf.* Figures 3.12a and 3.12b).



(a) COCO



(b) Human 3.6M

Figure 3.12: Skeleton structures used in COCO [25] (a) and Human3.6M [48] (b)

Therefore, in the simulation, a skeleton structure is used, which covers multiple skeleton structures used, for example, in the COCO [25] dataset, the Human3.6M dataset, and the MPII dataset [33] (see Figure 3.13).

Human joint positions are already available in the system due to the underlying skeleton of a 3D human model, and their position and orientation data can be read out correspondingly via the Unity game engine. However, depending on the

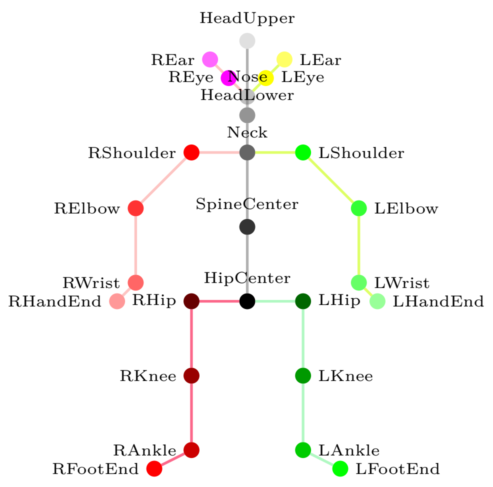


Figure 3.13: PedRec skeleton structure used in this work

source of the 3D models or the rigging method used, these skeletons may differ, and sometimes not all joints are available (for example, the position of the nose). These non-existing positions are calculated during the import of the human models based on the body proportions and stored as a virtual joint in the character model, allowing these positions to be retrieved. In each frame, all joint positions are queried for each 3D human model. Additionally, for each virtual camera, ray tracing is used to check whether there is an obstacle between the camera and the respective joints which obscures the joint. Since the camera positions and calibration data are known, the 3D position of the joints can be calculated for each camera in the respective camera coordinate system. The positions are then stored as ground truth for each frame together with boolean flags, indicating whether they are visible in the camera image and if an obstacle obscures them. Further, the orientation data of the head and the center of the hip are written out as quaternions. These data can be used as ground truth for the head and body orientation.

3.7.4.2 2D Joint Positions

The 2D joint positions are simply calculated from the 3D joint data by projecting it over the camera calibration data onto the 2D image plane. The x and y image coordinates are then written out as ground truth. Again, boolean flags indicate whether the human joints are visible on the camera image.

3.7.4.3 Bounding Boxes

Since this work focuses on the detection of pedestrians, mainly ground truth data for humans are output. This fact can be exploited to simplify the calculation of bounding boxes. For human avatars, the already available joint positions from the pose ground truth are used. The bounding boxes can be calculated from the minima and maxima of the x, y (and z in the 3D case) coordinates of the human joints. Additionally, a margin of 5% of each side length is added. This results in a small margin around the person, similar to manually annotated bounding boxes. Unity also uses bounding boxes internally, for example, to calculate collisions or to check whether an element should be rendered. These internal bounding boxes are in the world coordinate system and can therefore also be converted to camera coordinates. This method is used to calculate and output bounding boxes for non-human objects in the simulation. For bounding boxes, it is also indicated whether the bounding box is in the image or whether parts of the bounding box are hidden. The occlusion of human avatars is calculated based on the occlusion information of the joints; if $n \geq 1$ joints are occluded, the bounding box is also occluded. For other objects, a fine mesh grind of ray casts is used to check if there are objects between the camera and the object. Note that there is a possibility that there are smaller objects between the joints or between the ray cast grid for both methods. However, it is assumed that occlusion by small elements is trivial, and therefore, the object is considered unoccluded.

3.7.4.4 Actions

Actions like walking, looking for traffic, or kicking are annotated in the used motion capture files and provided as metadata. They are annotated with the frame rate in which the animation data is exported. Thus, the metadata contains the information on which actions were present at a frame for a given frame rate. The Unity game engine provides methods to get the current time t of a played animation in a normalized form between zero and one. The frame number at the animation clip frame rate has to be retrieved from this normalized time to get the actions in this frame. Let f_r be the frame rate of the animation clip and l the full length in seconds. The total number of frames t_f in the animation clip can be calculated as $t_f = f_r l$. The current frame number to query the action labels can be retrieved by $c_f = t_f t$.

3.7.4.5 Semantic Segmentation

Each element in the simulation has a label defining the segmentation class. The list of segmentation classes is equal to the list of classes used in the Cityscapes dataset [127, Table 8]. An additional virtual segmentation camera is created for each camera in the scene. In this camera, every element's shader is replaced by an unlit shader with a fixed surface color defined by the segmentation class. This unlit shader is unaffected by the lighting. Thus, always a fixed color is output. This segmentation image is saved in addition to the actual camera image and provides the segmentation class in color per pixel.

3.7.4.6 Depth

Unity provides access to the values from the depth buffer. As such, the depth ground truth can be directly taken from the depth buffer and is output as a depth image in addition to the actual virtual camera image.

3.7.4.7 Other Ground Truths

Various other possible ground truth data can be extracted from the simulation, but are not used in this work. For example, ground truth for ground contact of a person's feet can be easily retrieved by ray casting from the foot's end joint towards the floor and thus retrieving the current height above the floor for the foot. Another possible ground truth is the goal of a simulated person. This ground truth is available from the motion capture manager data in the form of the recorded participant's received instructions. Thus, there are many possibilities to generate additional ground truth for various tasks using a simulation.

Chapter 4

Applications

In the following, the applicability of simulation data for training deep neural networks will be verified through the use and development of recognition systems. Based on the results of previous work (see section 1.3), the following main hypotheses are posed for this purpose:

1. The use of simulation data for training deep neural networks for vision tasks, which receive purely visual sensor information as input, leads to overfitting on simulation data.

2. By using abstraction layers, such as human joint positions, instead of direct features from the visual input, action recognition systems can be trained with simulation data only.

3. Labels, that are not currently present in real datasets, can be created with the simulation framework and used to train DNNs to achieve functionality which would not be possible with current, freely available datasets.

These hypotheses have been tested in various projects to investigate the usefulness of simulation data in deep-learning applications. First, section 4.1 investigates how a deep-learning-based pose recognition algorithm, that has problems recognizing specific poses from corner cases, can be fine-tuned with simulation data. In section 4.2, a system for action recognition in a pedestrian parking lot situation was developed and enriched with simulation data. The action recognition experiments were continued in section 4.3 to investigate how the human abstraction '2D Human Pose' can be used to generalize from simulation data to real data. For this purpose, a system was trained purely on simulation data and evaluated on similar real data. Section 4.4 shows how simulation data can be used to train algorithms that required very complex labels. For this purpose, a system was developed that performs multiple perception tasks in a multitask approach and can be used to recognize 2D and 3D human poses as well as head and body orientation. Finally, in Section 4.5 it is shown that the 3D human pose data enables action recognition based on 3D joint information located in the local human coordinate system. This makes the system independent of camera movements. The system will be tested on a large simulation dataset specifically designed for pedestrian situations.

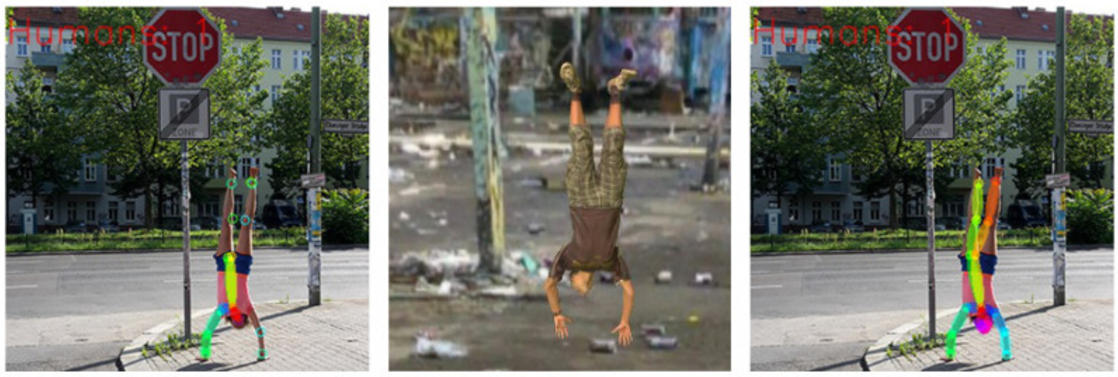


Figure 4.1: Results of OpenPose, trained on MS COCO, on real (**left**) and simulated (**center**) data as well as the result after fine-tuning on simulated data (**right**). Real image by M. Wildner (2017).

4.1 Improving the Detection of Corner Cases in 2D Human Pose Estimation

By using a simulation environment, content can be easily and quickly replaced to create a high level of data variation. This applies especially to most technical and behavioral parameters, which are hard to obtain in real-world scenarios and 3D human models. The ability to control these kinds of parameters is important because they influence the foreground appearance, of which a large number must exist to cover all visual characteristics of scenes, objects, and people. The focus of this application of simulated data is on corner cases in human pose estimation. There are successful algorithms available, which perform well in most cases. However, there are human movements that are not covered by standard datasets used to train these algorithms and to which they cannot generalize. There is a demand for the possibility to train specific problematic cases which are not recognized by algorithms without having to recreate a lot of real-world data for them. Examples are pedestrians overseen by autonomous cars or people who are undetected by infrastructure cameras of critical locations like train stations. It could be observed that a SOTA algorithm [27] for pose recognition had problems recognizing handstands, flips, and kicks (see for example Figure 4.1). As such, the focus of this work is on these action types to demonstrate and evaluate the simulation framework. Although these scenarios are often associated within sports venues, the ongoing public interest in parkour and the rising trend in urban yoga practice brings such human poses out onto urban sidewalks. Given the dynamic and unexpected nature of such human motions, autonomous driving systems must be able to perceive and respond to them, for although they are rare, they pose a critical threat that should not be overlooked.

It is investigated if and how well a SOTA algorithm can be trained to recognize corner cases by using only simulated data. The simulation framework presented in this work is suitable for use in various applications, including the autonomous driving context. For such a use-case, the first step towards developing algorithms, which allow vehicles to react to atypical pedestrian behavior, is to extract examples of such scenarios from vehicle sensors and public camera recordings. Afterward, simulation data for such human poses are created and tested on the autonomous recognition algorithms. Cases that proved challenging and in which the algorithms fail to recognize a pedestrian due to the obscurity of their pose are then used to train the algorithm in the next stage. In this way, the simulation framework allows for developing autonomous driving algorithms that react to obscure yet critical pedestrian poses by creating sufficiently sizeable simulated datasets of these poses with which the algorithms can be trained.

There is little literature addressing the possible poses of individuals in urban scenarios. Most of the work in this realm is focused on predicting the motion of a person or group of individuals. Völker & Kistemann observed the average number of persons per hour on a promenade in Cologne [128]. They tracked 1.838 pedestrians, 339 cyclists, 32 joggers, and Nordic walkers, as well as eight skaters. This example alone shows that skaters, joggers, and cyclists are considerably underrepresented in contrast to the average pedestrian. Autonomous systems should be able to recognize and interpret uncommon human motions correctly. An example of this is illustrated in Kidder's work on observing people who run through a parkour course in urban spaces [129]. Such activity includes many jumps and interactions with the environment, actions which, at present, are not found in datasets used to train algorithms for pedestrian recognition. Due to the low availability of such empirical data, alternative possibilities must be found to detect, record, and label relevant motions.

There has not yet been evidence showing that simulations improve the performance of human pose recognition for corner cases. The advantage of employing a new data generation and simulation framework for human poses and its benefit for retrained 2D human pose estimation is demonstrated.

The contribution in this application is to show if and how a well-working human pose estimation algorithm can be fine-tuned using only simulated data to recognize corner case poses that it could not detect when trained only on standard datasets.

4.1.1 Data Generation

In the experiments, 3D avatars are supposed to perform rare actions. Whole bodies without occlusions are considered. To generate the required data with the simulation framework (see chapter 3), 3D full-body scans as well as motion capture data were used. The 3D scans were generated in a laboratory at Reutlingen University. Motion capture data from the CMU motion capture database [130] was used,

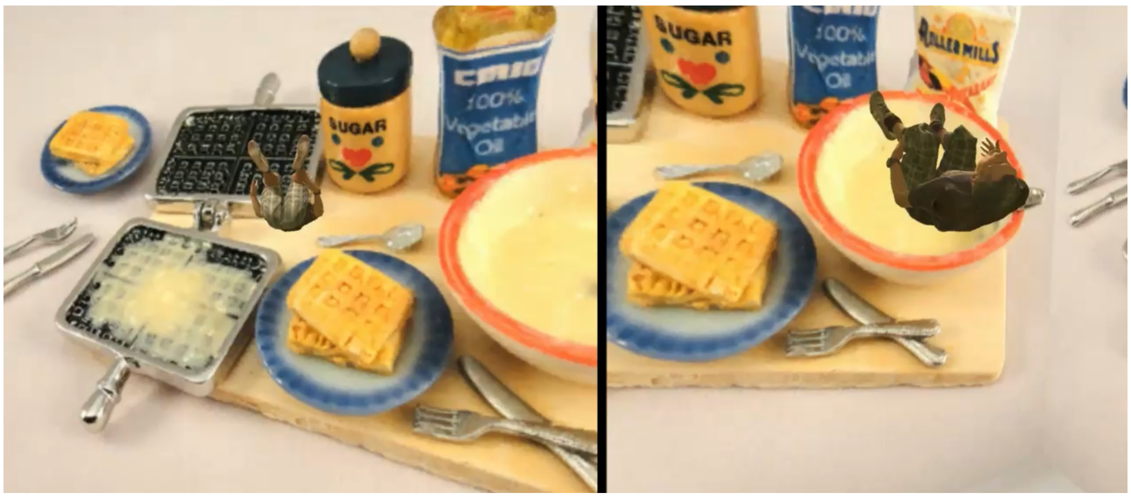


Figure 4.2: Example of the generated training data. View from two perspectives on a 3D avatar doing a backflip in front of a random background.

which already contains sequences of handstands, kicks, and flips. An example of the sensor data generated can be seen in Figure 4.2.

4.1.2 Experiments

The experiments show how well algorithms, which are fine-tuned on simulated data, perform on data with which they had issues before. A small evaluation dataset - RARESIM - was created, containing people doing handstands, flips, and kicks. In total, 44 real-world pictures were collected and annotated manually. Some pictures are shown in Figure 4.7. Further, two different simulated datasets were used for training. One rather small dataset - SIM - contained $36,225$ images collected from 35 camera positions ($1,035$ images from each camera). Every image contained one 3D model, out of only two, performing one of the actions handstand, flip or kick. The background was changed only on a change of the replayed motion. The models were trained together with the full COCO 2017 training dataset, which contains $123,758$ images, leading to a simulated to real image ratio of 0.29 to 1. Second, a bigger simulated dataset - BIG-SIM - was created. In this dataset, the same motion capture data as in SIM were used, but with more repetitions, different backgrounds for each frame, ten foreground appearances, and 48 cameras. It contains $6,880$ images per camera, leading to a dataset size of $453,998$ images in total with a simulated to real image ratio of 3.6 to 1.

Table 4.1: PCK results on the RARESIM dataset.

Method	Ears	Eyes	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Total
baseline	39.1	46.7	55.4	51.1	48.9	48.9	30.4	29.3	28.3	42.0
SIM	56.5	71.7	69.6	64.1	**62.0**	**63.0**	41.3	38.0	33.7	55.6
SIM-fixed	53.3	67.4	**73.9**	60.9	56.5	55.4	**46.7**	39.1	**40.2**	54.8
BIG-SIM	**58.7**	70.7	67.4	**65.2**	57.6	59.8	40.2	**42.4**	39.1	**55.7**
BIG-SIM-fixed	57.6	**72.8**	65.2	56.5	52.2	54.3	42.4	39.1	**40.2**	53.4

4.1.2.1 Algorithm

For the experiments, a custom implementation[1] of the OpenPose pose estimation algorithm proposed by Cao *et al.* [27] was used. The network was reimplemented in PyTorch and the weights of the trained CMU model were used as a baseline model. The CMU model was trained on the COCO dataset, containing around $250,000$ people annotated with keypoints. The model achieves SOTA performance with an AP of 60.5% on the COCO 2017 keypoint test challenge, as well as a mAP of 75.6% on the MPII full testing set. It also offers real-time performance on GPUs, making it interesting for applications in autonomous systems and public space surveillance. As the authors state in their paper [27], the most common failure causes include rare pose or appearance, as well as missing or false part detections.

The network model outputs heatmaps for joint positions as well as part affinity fields which encode the location and orientation of limbs (see 4.3). The first part of the network structure is a feature extractor F, for which the first ten layers of VGG-19 are used. F was trained on ImageNet and fine-tuned during the CMUs training process. After the feature extractor, a six-stage approach is used for extracting the joint maps S^t as well as the part affinity fields L^t where t stands for stage one to six.

$$S^1 = \rho^1(F) \tag{4.1}$$

$$L^1 = \varphi^1(F) \tag{4.2}$$

Stage one is the bootstrap stage and uses the image features computed by F to produce first proposals for the joint map S^1 and the part affinity field L^1 (c.f. Equation 4.1 and 4.2). ρ^t and φ^t denote the network layers for inference at stage t, which have the same structure for each stage.

$$S^t = \rho^t(F, S^{t-1}, L^{t-1}), \forall t \geq 2 \tag{4.3}$$

$$L^t = \varphi^t(F, S^{t-1}, L^{t-1}), \forall t \geq 2 \tag{4.4}$$

On stages $t \geq 2$ the outputs S^t and L^t of each stage are used together with the image features from F as input for the following stage in a recursive manner.

[1]https://github.com/noboevbo/openpose-pytorch

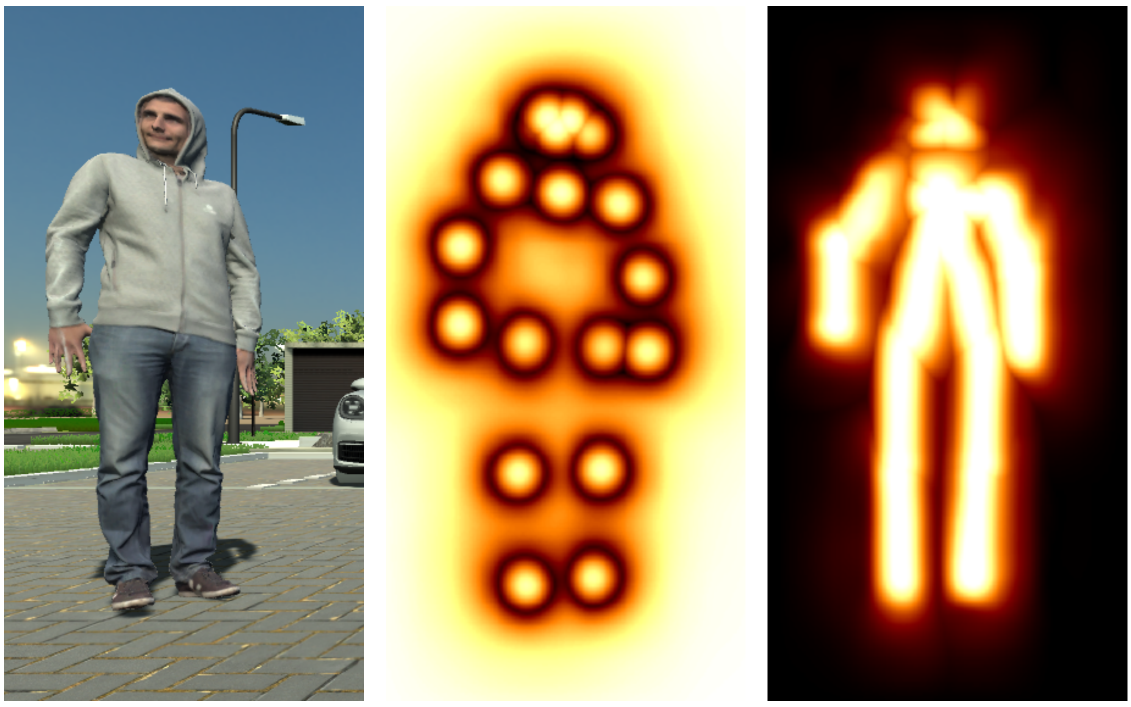

(a) Example Input (b) Joint Heatmaps (c) Part Affinity Fields

Figure 4.3: Open pose network outputs. Based on input a the network outputs joint heatmaps b as well as part affinity maps c from which the joint positions are derived in post-processing.

Most of the descriptors which are directly dependent on the visual appearance are hypothetically contained in the feature extractor F as well as in the bootstrap phase S^1 and L^1.

At the current stage, the focus of this work is on rare poses and on false or missing part detections caused by the pose rarity. In the 'fixed' experiments, the layers F, S^1 and L^1, which probably rely primarily on visual appearances and only train later stages, which hypothetically use more abstract features, were frozen. This kind of training is intended to not disrupt descriptors for visual appearances in the real world by using simulated data, assuming a considerable domain shift between real and simulated data. The abstract features from layers $t \geq 2$ should be sufficient to learn new skeleton configurations with the existing visual descriptors trained on the huge real-world datasets ImageNet [16] and COCO [25].

The L2 loss function and the stochastic gradient descent optimizer with parameters learning rate 0.0008, momentum 0.9, and weight decay 0.0005 were used for training. A batch size of ten for training on an NVIDIA Titan X (12 GB Memory) were used. The evaluation was done using a concatenation of the original image results with scales 0.5, 1.0, 1.5, and 2.0, which leads to a better recognition rate.

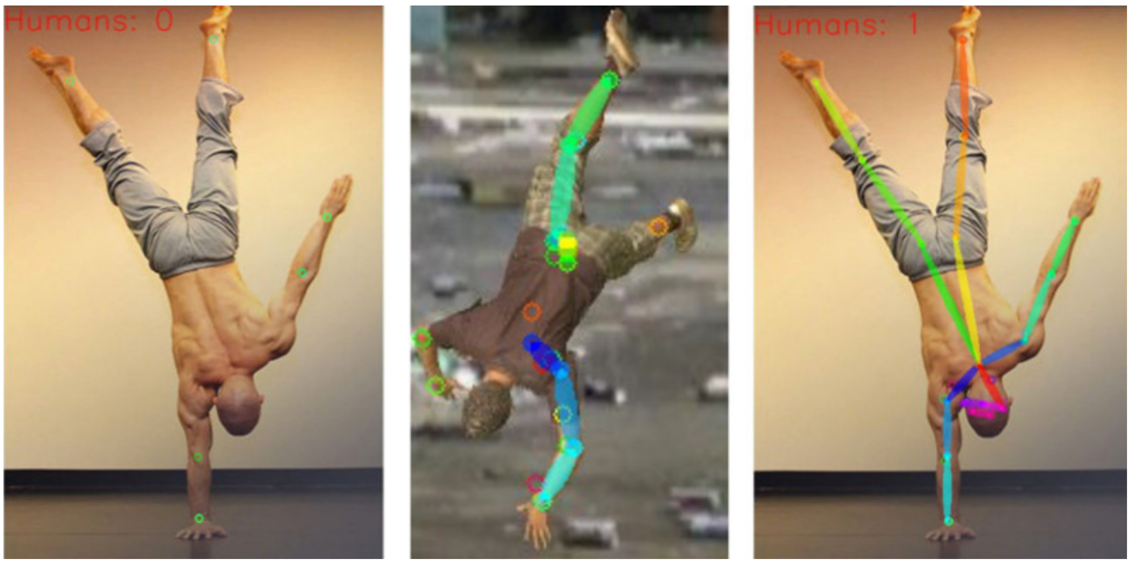


Figure 4.4: Example of training improvements. (left) results trained on COCO only (middle) results trained on COCO, used as training data (right) results after training on SIM. Real image by Y. Ayalon (2014).

4.1.2.2 Metric

The PCK metric (see section 2.3.3.1) was used to compare the performance of the different model states. As a second consideration on the performance of a model, it should be considered if the performance on the more general datasets decreases when the model is fine-tuned to corner cases via simulation. For this consideration, the OKS based evaluation metric was used that has been proposed for the validation dataset of COCO 2017 [25] (see section 2.3.3.3).

4.1.2.3 Results

The models are based on the OpenPose implementation using pre-trained weights from the original OpenPose caffe model. All models were trained for 700.000 iterations. The following list shows on which datasets the evaluated models were trained on, as well as if and which of their layers were fixed during training:

- **baseline**: COCO 2017
- **SIM**: COCO 2017 and SIM
- **SIM-fixed**: COCO 2017 and SIM, fixed layers F, S^1, L^1
- **BIG-SIM**: COCO 2017 and BIG-SIM
- **BIG-SIM-fixed**: COCO 2017 and BIG-SIM, fixed layers F, S^1, L^1

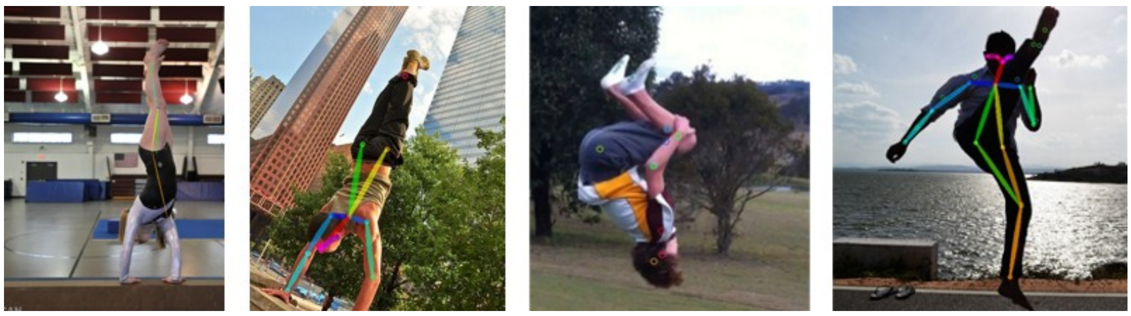


Figure 4.5: Examples from the dataset, which are not recognized or still contain errors, after training on simulated data. From left to right (1) only very few human joints recognized (2) missing human joints / limbs (3) no limbs detected / bad joint proposals (4) left / right confused. Real images (1-3) by X. Navarro (2014), P. Dep (2009) and Wikimedia (2013).

As seen in Table 4.1 as well as in Figure 4.6 the overall performance of the OpenPose algorithm increases on the small corner case dataset by 13.7% on the BIG-SIM model and 13.6% on the SIM model. The fixed models performed a little worse, but still improved the performance by over 11%. Especially the poses which are near to sensor information included in the simulated dataset, and are not recognized by the algorithm trained on COCO only, seem to be greatly improved (see Figure 4.4). There is also an improvement on missing parts caused by rare poses (*cf.* the kick in Figure 4.7 (lower right)). The performance on the COCO 2017 validation dataset is mostly constant for the baseline and the SIM models, but drops for the BIG-SIM models by 3.5% for BIG-SIM and 2.1% for BIG-SIM-fixed. Having such a big portion of specialized poses in the simulated training data leads to overfitting the model towards simulated data, decreasing performance on real-world data. Thus, using too big of a simulated data portion introduces a heavy bias towards simulated data, causing severe domain transfer problems, regardless of fixing layers. This leads to the conclusion that the OpenPose algorithm is also heavily dependent on visual appearance at later stages. Nevertheless, a remarkable improvement of 13.6% was achieved with the SIM model on the RARE dataset while keeping comparable performance on COCO (see Table 4.2), hinting that simulated data can deliver benefits in teaching algorithms to recognize corner cases.

As seen in Figure 4.5, there are cases that are not recognized even after training on simulation, which may be caused by simulation data being too different to the poses in these images or requiring better visual feature descriptors. It may be possible to gain those visual feature descriptors, creating more variability by including more 3D human models while generating simulated training data.

Table 4.2: Performance on the COCO 2017 validation dataset. AP^{50} is for AP@IoU=0.50 (all person scales). AP^{M} is for medium scale persons.

Method	AP	AP^{50}	AP^{75}	AP^{M}	AP^{L}
baseline	**56.8**	78.5	**60.9**	**54.7**	62.5
SIM	56.6	**78.6**	**60.9**	54.1	**62.6**
SIM-fixed	56.4	**78.6**	60.0	54.2	62.5
BIG-SIM	53.3	76.4	56.4	51.2	59.6
BIG-SIM-fixed	54.7	77.4	58.6	52.7	60.1

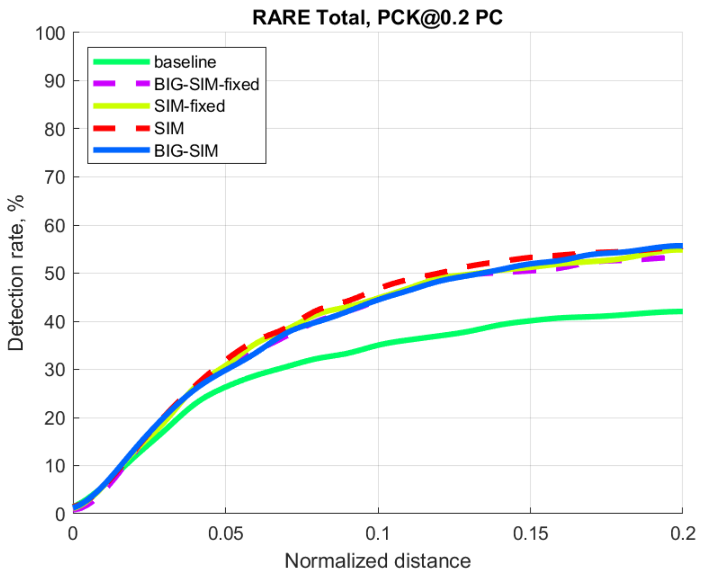


Figure 4.6: PCK result plot on the COCO 2017 challenge using the OpenPose algorithm trained on COCO only and COCO fine-tuned with specific simulated data.

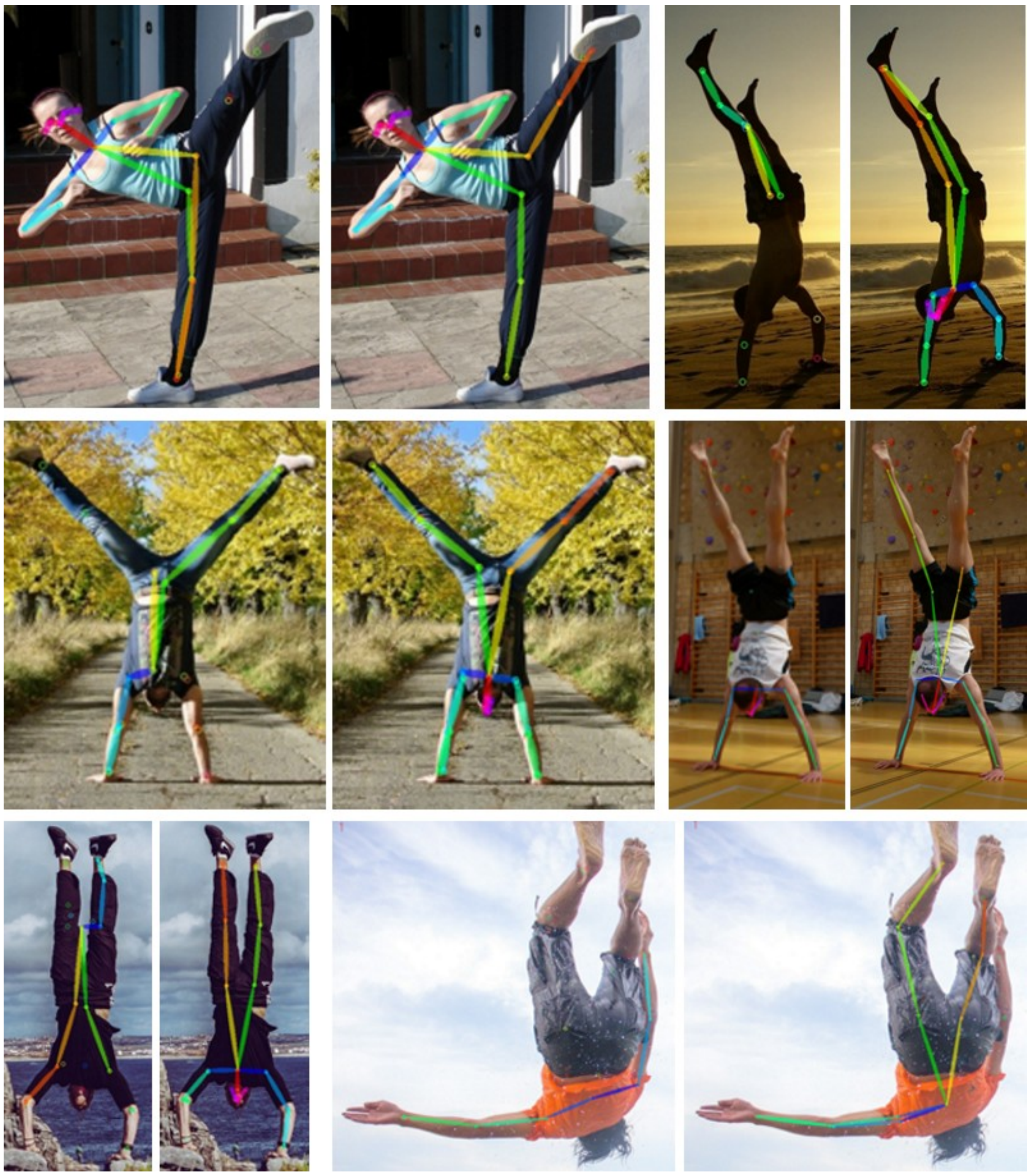


Figure 4.7: Examples of improved pose recognition on the dataset after training on simulated data (SIM). left is before training, right after training. Real image (upper left) by D. Shawn (2007).

4.1.3 Discussion

The application of the simulation framework demonstrated the possibility to acquire relevant data for the design and systematic test of camera perception systems. The usage of virtual sensor data for the training of algorithms based on the example of a SOTA pose estimation algorithm was evaluated. By fine-tuning the model with simulated data only, a significant detection accuracy improvement on a dataset mainly containing poses that caused false joint prediction has been demonstrated. The results show that it is beneficial to use simulations at least in addition to real-world data to train poses that are rare in real-world datasets. Furthermore, the experiments hint, that hypothetically the same kind of problems that cause trouble in predictions in real-world images exist in simulated data. This observation is investigated further, as it motivates the potential increased reliance on simulated data to evaluate algorithm performance at large scale. The following chapters will therefore examine in more detail how systems can be trained purely with simulated data and whether the results are applicable to real data.

4.2 Pose-based Action Recognition

As a continuation of the application of simulated data on corner cases (see section 4.1), there is great potential in the application of simulated data for the training of action detection algorithms. Yet, the experiments showed a domain shift from simulated visual data to real data. As stated in hypothesis two, an abstraction from the actual visual sensor data might help to overcome such domain transfer issues. Thus, in this part of the thesis, it will be demonstrated how human pose can be used as an abstraction layer for visual sensor input in human action recognition. One focus is on the development of a suited 2D human pose estimation pipeline, that includes tracking to provide temporal human pose data as input for the human action recognition. Another focus is on the human action recognition system, and first experiments with simulated and real training data. The developed pipeline will be modular to enable adaptability for further applications and experiments.

This application of the simulation framework as well as the development of the pose-based action recognition approach was part of the project 'Open Fusion Platform'[2], which was centered around an autonomous vehicle with a valet parking function. It should automatically search for a free parking space on a parking lot and automatically be able to drive back to a pick-up point. Pedestrians can be present in the parking lot. Thus, it is important to recognize them. In addition to the pure recognition of pedestrians, it is also important to recognize what they are doing. In the use case, they can be in front of the parked vehicle, as long as the vehicle is not moving. To drive off while a pedestrian is detected in front of the vehicle, the pedestrian must indicate, by a waving gesture, that the vehicle is allowed to drive out. Thus, the pedestrians have to be detected, and further, their current actions have to be classified. For the parking lot use case, the actions 'idle', 'walk' and 'wave' were specified to be detected.

The contributions in this part of the work are:

1. A recognition pipeline, which operates on 2D monocular camera images in real-time. It contains functionality to detect objects, humans, and their poses and track and estimate humans and their actions.

2. A pose-based action recognition algorithm, based on Encoded Human Pose Image (EHPI) input data as shown in figure 4.8, with SOTA performance on sensor-based pose recognition data.

3. A demonstration on how to improve the action recognition algorithm with simulated data.

[2]https://www.hella.com/ofp-project/de/index.html (accessed on 2022-03-06)

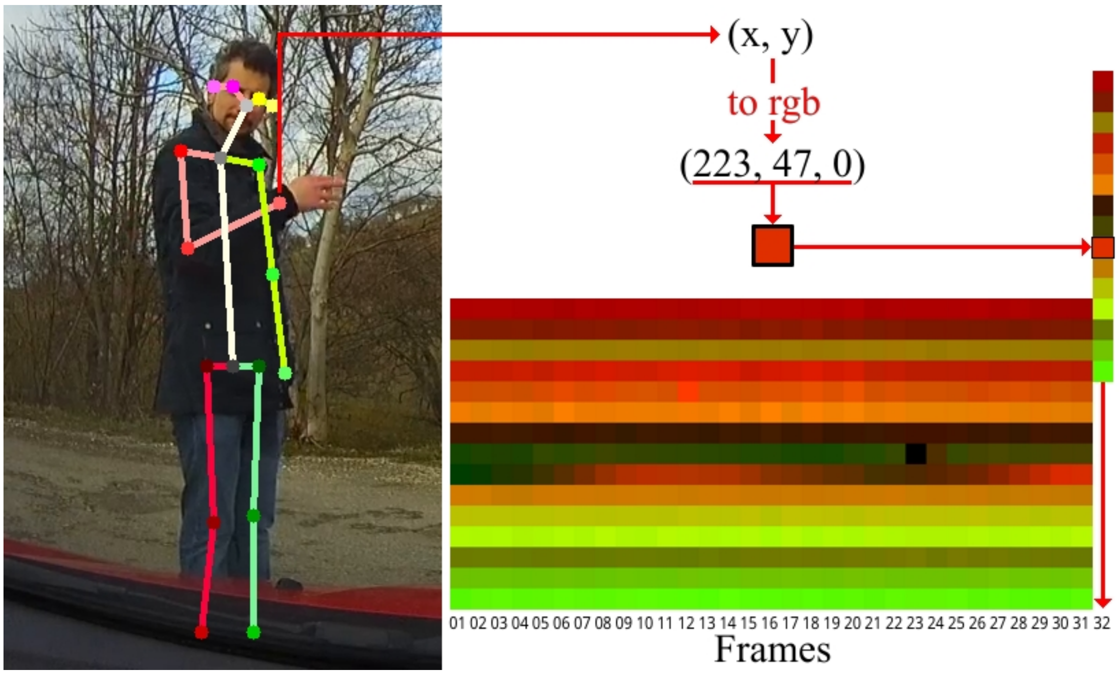


Figure 4.8: From skeletal joints to an EHPI, exemplified by the right wrist. The x and y coordinates are normalized (for the visualization the normalization process is simplified and ranges between 0 and 255 for RGB values), afterwards the x value is used as the red component and the y value as the green component. This RGB value is set for each frame in an n-dimensional vector at a fixed location. In the example $n = 15$ joints are used, the right wrist is set in row 9. The full EHPI is of size $32 \times 15 \times 3$.

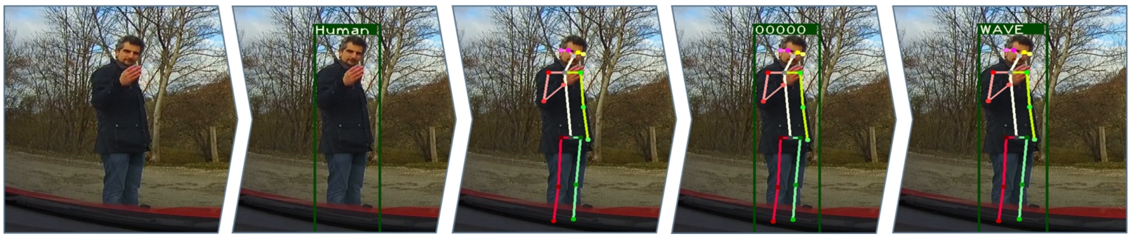


Figure 4.9: Real-time action recognition pipeline from a monocular camera mounted in the car observing the gesture of a potential user of the autonomous vehicle. From left to right: 1) Raw camera image, 2) Object detection [131], 3) Pose Recognition [30], 4) Pose-based human tracking and 5) Pose-based action recognition

4.2.1 Recognition Pipeline

All steps in the pipeline[3] are shown in Figure 4.9. In the first step, objects are recognized in a 2D camera image, especially people (see section 4.2.1.1), resulting in bounding boxes around the respective objects. If a person is recognized in a frame, a pose recognition algorithm is applied to the image information within that person's bounding box (see section 4.2.1.2). A top-down pose recognition algorithm was used, thus pose recognition has to be done for every human in the image. Based on the human poses in successive frames, a pose-based tracking (see section 4.2.1.3) of humans was developed. The action recognition is performed on these tracked human poses (see section 4.2.2).

This modular approach allows the more complex algorithms, such as pose recognition, to be applied only specifically. For example, pose recognition could only be applied, if a person is close to the autonomous system. This also allows using different training data at different steps and with different levels of abstraction, which in turn enables one to train the action recognition algorithm with simulated data (see section 4.2.3.1).

4.2.1.1 Object Detection

An object detection algorithm is used to obtain an initial estimate of humans' presence in the image. In addition to the algorithm's accuracy, the running time is the main criterion for selecting the object detection algorithm. Possible false detections of the object detection algorithm can be compensated by the pose estimation and the tracking of humans (see section 4.2.1.3). In the developed pipeline, Yolo V3 [131] is used as a compromise between runtime and accuracy, which was pre-trained on ImageNet [16] and the COCO [25] dataset. The object detection algorithm can be replaced by alternative object detection algorithms, depending on

[3]Code available at https://github.com/noboevbo/ehpi_action_recognition

accuracy and runtime requirements. The input into the object detector is an RGB camera image, which may be scaled down to allow faster processing. The algorithm then estimates possible object locations in the image in the form of bounding boxes and classifies their content. After post-processing, the resulting data is a list of classified bounding boxes. In this work, only bounding boxes are used, that have been classified as humans.

4.2.1.2 Pose Estimation

The approach from Xiao *et al.* [30] is used for human pose estimation, with its network pre-trained on the COCO [25] and MPII [33] datasets. The algorithm requires human bounding boxes as input and estimates a human skeleton in this cropped region. Like in most SOTA pose recognition algorithms, a heat map is predicted for each joint, indicating the estimated probability for each joint. During the post-processing, non-maximum suppression is performed, and a human skeleton is reconstructed in the form of 2D joint positions and their connections. In comparison to OpenPose, used in the experiments shown in section 4.1, this algorithm has a much simpler architecture, is faster and, as a top-down approach, is more suited for a modular system.

4.2.1.3 Human Tracking

For the action recognition algorithm, a person's skeletal information is required across multiple frames. Since the pose recognition algorithm described above is applied to single images, the skeletons in several frames are initially independent. Skeletons are tracked based on their joint positions to establish the reference of skeletons across several frames. The pyramidal implementation of the Lukas Kanade Feature Tracker [132] is used with the joint positions of the human skeletons in the image as features to be tracked. This results in an estimated skeleton in frame n for each skeleton in frame $n-1$. With these tracked skeletons from frame $n-1$ and newly detected skeletons from frame n, several skeleton proposals exist, which need to be merged as follows.

A merge is done by measuring the similarity of two human skeletons. If they are sufficiently similar, the two skeletons will be merged into one human skeleton, thus tracking the human over time. In addition to comparing detected and tracked people for a possible merge, all detected people must also be compared. The same person could be detected several times by false detection of the object or pose recognition algorithm.

The first step to merging two human skeletons is to find the similarity between them. Let a and b be human skeleton hypotheses from a list of detected or tracked skeletons. Δ_a is defined as the maximum distance between joint i in two skeletons to be considered part of the same skeleton. Δ_a is calculated by using the bounding box width w_a and height h_a of human a (see equation 4.5).

$$\Delta_a = F(\sqrt{w_a^2 + h_a^2}) \tag{4.5}$$

Factor F denotes a hyperparameter, which corresponds to the percentage of the human's bounding box diagonal. Setting $F = 0.025$ has proven good in practice. Subsequently, the Euclidean distance δ_{ab_i} of joint i between human skeleton a and b is calculated (see equation 4.6).

$$\delta_{ab_i} = \|a_i - b_i\|_2 \tag{4.6}$$

This distance is only considered if a and b contain joint i with a minimum probability of T_J that specifies the minimum joint quality required to use joints in the tracking process. Setting $T_J = 0.4$ worked well in practice. This constraint is included to enable tracking even when some joints are not recognized or only poorly recognized, for example, due to occlusion. Then the similarity score (S_{ab_i}) is calculated for joint i by comparing the actual joint distance with the maximum acceptable distance (see equation 4.7).

$$S_{ab_i} = \begin{cases} 1 - (\delta_{ab_i}/\Delta_a), & \text{if } \delta_{ab_i} < \Delta_a \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

The similarity (S_{ab}) between human skeleton a and b is calculated by combining all joint similarities S_{ab_i} (see section 4.8).

$$S_{ab} = \frac{1}{I} \sum_{i=1}^{I} s_{ab_i} \tag{4.8}$$

Factor I denotes the number of joints used for tracking. Afterward, all detected human skeletons are tried to be merged with other detected human skeletons to avoid repeated recognition of skeletons belonging to the same person. A threshold of $T_S = 0.15$ is defined to specify when two human skeletons are similar. If the similarity score is above T_S for two detected humans, the detection with the lower score is removed from the detection list. After merging the detected human skeletons, they are merged with all tracked humans from previous frames. In this merge process, every detected human skeleton is compared to every tracked human skeleton. Suppose the similarity score of two skeletons is above T_S. In that case, the identifier of the tracked skeleton is assigned to the detected skeleton, and the tracked skeleton is removed from the tracking list. Further, suppose that some tracked human skeletons are leftover after the merge process, meaning that the object detector did not provide a human proposal at the location of a tracked human skeleton. In that case, the pose estimation is applied on the bounding box of the tracked human, and if a human skeleton with a score higher than T_J is estimated, this human will be kept with its identifier as a detected human. With this approach, false negatives from the object detector can be compensated, and it allows the deactivation of the object detection to improve performance (see section 4.2.1.4).

4.2.1.4 Runtime Performance

The runtime of the pipeline scales with the number of people to be detected. Usually, only the humans in the immediate surrounding area of the autonomous system are relevant, so the entire pipeline may typically not have to be used for all humans in the image. For one human with an input image resolution of 1280x720, downscaled for processing to 640x360, the entire pipeline runs on average with 29 FPS. For two people, the FPS is reduced to around 21 FPS, which still ensures real-time processing. To improve performance in exceptional cases, object detection, usually performed for each frame, can be disabled once a person has been detected. The pose and action detection can then be continued for this person based on bounding box proposals from the tracking process. On average, by switching off the object detection, 57 FPS can be achieved for one person. Depending on the requirements, it would be possible to perform object detection only on a limited number of frames. It is also important to note that the implementation is not designed for performance, as more emphasis was placed on code readability. It can be assumed that the performance can be increased with appropriate adaptations. All performance tests were carried out on a laptop with an Intel i7-8700 six-core CPU and an NVIDIA GTX 1080 GPU using Ubuntu 18.04 with CUDA 10.0 and CUDNN 7.4.2.

4.2.2 Pose-based Action Recognition

Since a substantial amount of progress has been made in the field of convolutional neural networks, it was decided to investigate an approach in which human skeletons are encoded overtime in an image-like data structure.

The basic process is shown above in Figure 4.8. Once the pose of a human has been extracted from the camera image, the basic idea is to encode the x, y, and z positions of the joints as red, green, and blue values in an RGB image. In this work, 2D human pose estimation is applied on monocular camera images, so the z value is not used, and thus the blue channel is set to zero. The channel could also be removed as long as only the x and y coordinates are used. To convert the global joint coordinates into corresponding 'color values', they are normalized. This process is described in more detail in section 4.2.2.2. Note that the values are normalized as network input in the continuous range from zero to one and not as discrete integer values from ranging from zero to 255. Thus, the analogy of an image is therefore not entirely accurate. However, for visualization purposes, the joint positions are normalized between zero and 255 for the figures. Any number of joints can be encoded. In the current work, the nose, neck, pelvis, left shoulder,

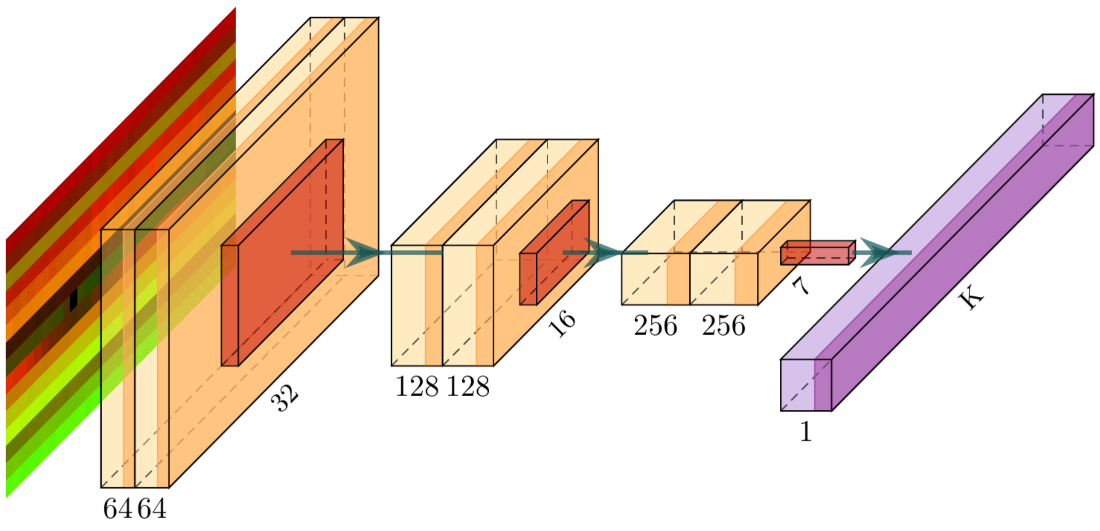


Figure 4.10: Simple network architecture that is used to classify the EHPIs on the JHMDB dataset.

left elbow, left wrist, right shoulder, right elbow, right wrist, left hip, left knee, left ankle, right hip, right knee, and right ankle are used in this particular order. The encoded joints are assigned in a fixed order in a $1 \times n \times 3$ matrix, where n stands for the number of joints. After the human pose for a frame has been encoded into such a matrix, it is appended as the last column to a $m \times n \times 3$ matrix and replaces the first column of this matrix if it already contains m frames. Each column represents an encoded human pose in a frame. The full matrix represents an Encoded Human Pose Image. In the current work, $m = 32$ is used because about one to two seconds of movements should be analyzed to give an action estimate. Additionally, it corresponds to the standard image width used in machine learning applications. The recognition results of the last 20 frames are considered, and the action class with the highest summed probability in the last 20 frames is used as the prediction to stabilize the action recognition.

4.2.2.1 Network

For the classification of the EHPIs straightforward network is used. It consists of six convolutional layers, of which each has a 3×3 kernel and both padding and stride of one. A fully connected layer is placed at the end for the final action classification (see Figure 4.10). Each convolutional layer is followed by batch normalization [110]. As an activation function, ReLU is used in the convolutional layers. After the second and fourth convolution, a max-pooling layer with a kernel size of 2×2 reduces the spatial resolution by factor two. After the last convolutional layer, a global average pooling layer is applied. Xavier initialization is used [104] for all convolutional layers. The deeper the network is, the more spatio-temporal context should be encoded in the learned features due to larger receptive fields.

Since the use case contains considerably more data than the JHMDB [81] dataset contains (see section 4.2.3), the network is no longer sufficient. Expanding the network with further convolutional layers and also increasing the size of the fully connected layer would result in the network having more parameters than some existing and efficient CNNs for classification. Therefore, ShuffleNet v2 [133] architecture was applied, with which also the application of standard computer vision algorithms to EHPIs is demonstrated.

4.2.2.2 Preprocessing of EHPIs

The normalization of the EHPI takes place on the entire $m \times n \times 3$ matrix. The encoded x and y joint positions are normalized independently between zero and one. This type of normalization is intended to ensure the independence of the body size of different people while maintaining the relative change in scale through a different distance to the camera. The local range of motion for a time window of length m is considered correspondingly. Before normalization, human body joints outside the image are removed as a preprocessing step by setting their coordinates to zero. When joints are not recognized or have a probability below T_J, the x and y values for them are set to zero. The same applies to human poses, which are not recognized at all. In this case, the complete $1 \times n \times 3$ matrix is set to zero. An EHPI requires at least two frames with human poses to be considered.

4.2.2.3 Example EHPIs

Figure 4.11 shows the EHPIs and a camera image of the last frame (rightmost column) of the EHPI for three examples of the actions ‘idle’, ‘walk’ and ‘wave’. The row representing the joint of the right wrist is plotted in an enlarged view, since it is diagnostic for describing the actions of interest in the following example. For the action ‘idle’ the color representation is relatively constant over the whole period because there is hardly any movement of the joint. For the action ‘walk’, one can notice a smooth transition from green to orange. This is because the joint of the right wrist moves from left to right of the image during the EHPI period. Therefore, the normalized x value increases towards one (in the visualization, thus the red value towards 255), while the y value (in the visualization, the green value) remains relatively constant. During the ‘wave’ action, one can notice a repetitive color gradient from green to red because the joint of the right hand repeatedly moves in the x direction during the wave movement.

Figure 4.12 shows a sequence of a wave movement in more detail. At the end (right) part of the EHPI for the joint of the right wrist, the color encoding gets redder when the joint is on the right side of the picture. Further, the effects of false

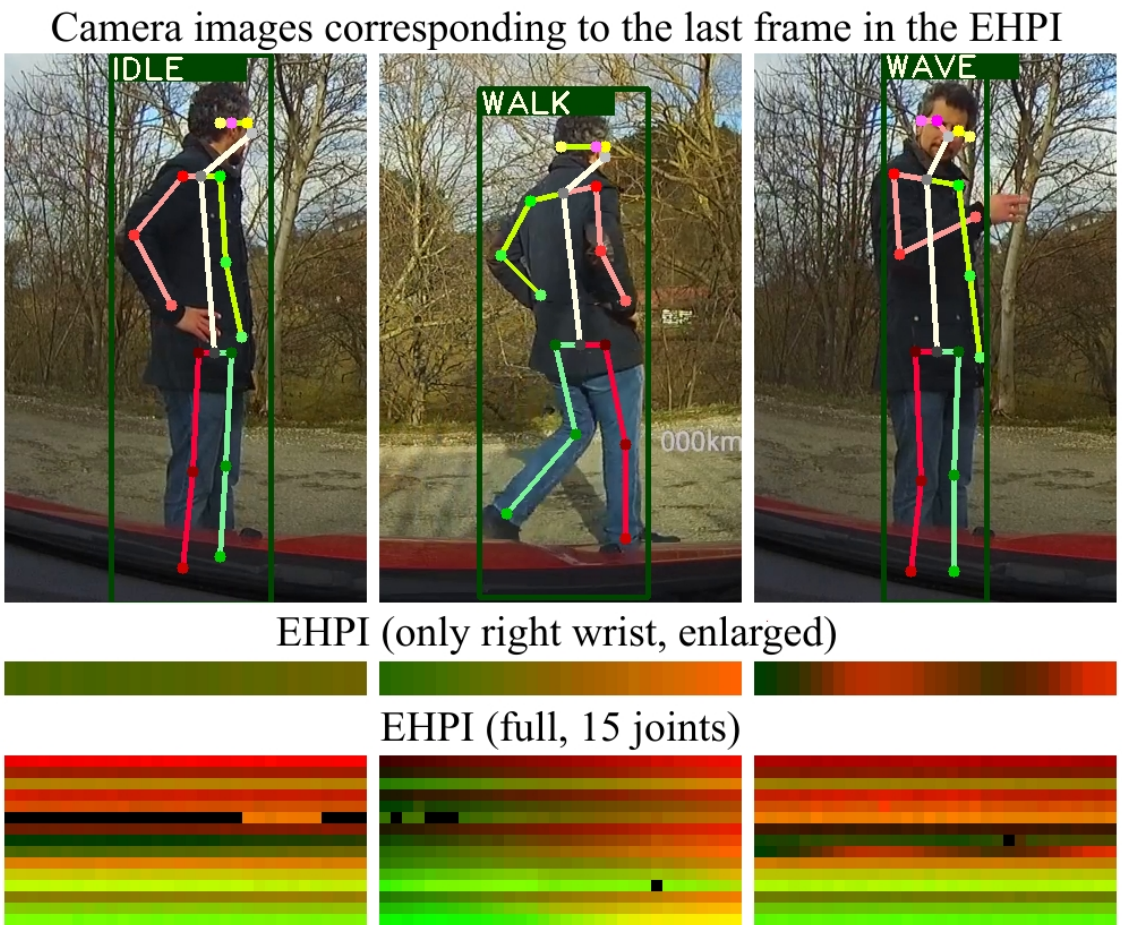

Camera images corresponding to the last frame in the EHPI
IDLE
WALK
WAVE
000km
EHPI (only right wrist, enlarged)
EHPI (full, 15 joints)

Figure 4.11: EHPI examples of different actions. The example of the right wrist, which is explicitly shown at three times its height, clearly shows that a smooth color gradient is visible in the idle action, a color gradient from green to orange is visible during walking and a repetitive gradient from green to red is observable during waving.

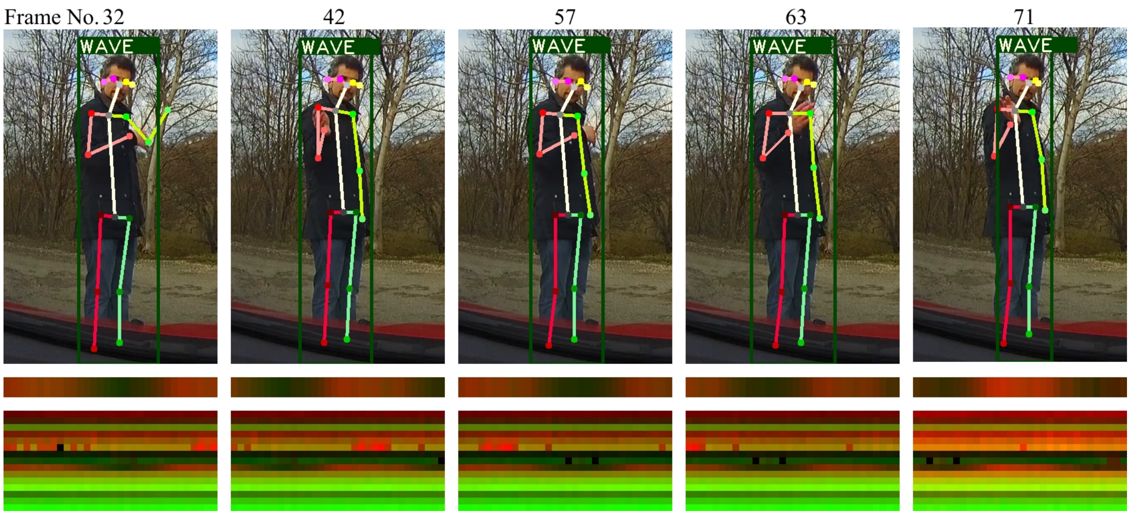


Figure 4.12: Five frames from a sequence with camera image, EHPI (right wrist, enlarged) and the full EHPI. The EHPI for the right wrist moves during waving towards red (maximum value in x direction). The first picture shows false detection of the left wrist (EHPI, row 6), which is filtered by the application of noisy training data of the action detection. In the last image, the whole EHPI is shifted more into the red. This is because there are no more extreme false detections of the left wrist that shift the maximum x value during normalization.

detection of body joints are displayed clearly. In this example, the left ankle (row six in EHPI) is partially recognized incorrectly and thus is encoded much further to the right than it actually is. This becomes clear with the strongly red coded areas, which appear without a clean transition from green to red. The joint of the left ankle was recognized with a probability above T_J by the pose estimation algorithm. Thus, it is not encoded as zero. Due to the distorted, more extreme red values of the left hand, the entire EHPI is shifted a little into the green area, which becomes clear in the last image (frame 71), where most of the joint recognition errors are no longer present. The color of the entire EHPI shifts towards red.

4.2.3 Datasets

The JHMDB [81] dataset consists of 928 videos, of which each has an annotation label denoting one of 21 action classes. Each video has a resolution of 320×240 pixels. The evaluation is done on three splits, of which each uses about 30% test data. Results are reported as the overall mean splits. From here on, JHMDB refers to the entire dataset, while JHMDB-1 refers to JHMDB split 1 with pose data from the pipeline. JHMDB-1-GT refers to JHMDB-1 with pose data from the JHMDB ground truth.

The automotive parking lot use case dataset SIM consists of various camera sequences. Different videos were recorded with a Logitech C920 webcam, an iTracker GS6000 dashcam, and a Yi 4k+ camera. It is a very use case-specific dataset, which only contains the actions 'idle', 'walk' and 'wave'. Recordings were partly taken inside buildings, partly also in use case situations in the vehicle. In addition, the dataset contains some simulated elements, which are described in more detail in section 4.2.3.1. The entire dataset consists of 216 labeled sequences and a total of $61,826$ EHPIs. All videos have a resolution of 1280×720 at 30 FPS. All sequences contain actions from the same person. For the evaluation, 27 sequences with a total of $8,351$ frames are used. All sequences are cuts from one scene, which corresponds to the use case. The scene was recorded simultaneously from the dashcam and the action cam to get data from two different sensors in slightly different locations.

4.2.3.1 Simulated Data

The experiments in section 4.1 have shown that the use of simulation can significantly improve the performance of pose recognition algorithms. Thus, simulation data is also used in this experiment to enrich training data further. The advantage of the modular pipeline is that simulation data can be used as training data at different steps in the pipeline, while real data is used at other steps. In action detection, this offers the great advantage of having the abstraction layer of the pose data between the sensor information and action detection. The underlying hypothesis is that this abstraction layer prevents sensor domain transfer problems between simulation and real data. In principle, motion capture data alone is sufficient to generate ground truth data for the pose-based action detection. By a corresponding 2D projection of the 3D joint coordinates for any number of camera positions in 3D space, 2D human pose information can be obtained without generating camera sensor simulations. Since pose recognition algorithms are not perfect and artifacts like a slight jittering of the joint positions, false recognition of joints or not recognizing joints can occur, additionally simulated camera images are generated to apply the pose recognition algorithms and use the output as ground truth with such kind of natural noise for the action recognition. In previous work, evidence was provided that pose recognition algorithms have similar problems on simulated data as on real data [5]. Thus, the same bias on estimated human poses on simulation data as on real data is expected. The motion data of the actions 'idle', 'walk' and 'wave' were recorded in our motion capture laboratory. One person performed every action ten times for ten seconds. The motion data is used to animate a 3D scanned human model in the Unity-based simulation. A flat area with a skybox for the background without any other environmental details is used as the environment (see Figure 4.13). Finally, various virtual camera sensors can be placed around the person, generating corresponding sensor information. For each virtual camera image, the ground truth, in this case, the 2D human pose and the corresponding action, can be generated automatically [5]. Figure 4.13 shows some examples of simulated sensor information. A total of six camera positions is used in this work.

SIM (gt) contains the perfect pose data from motion capturing directly, SIM (pose) contains the pose data from the output of the pose estimation pipeline, and SIM contains the data from both sources.

4.2.3.2 Data Augmentation

Augmentation is applied to increase the variance in the training data. Joints are flipped horizontally in 50% of the cases. If the image is flipped, in 50% of the flipped images, the indexes of the left and right joints are also switched, thus in 25% of the cases, a person looking in the other camera direction than originally is simulated. In addition, joints are partly removed to simulate occlusion. In 25% of the cases, the joints of both feet and in 6.25% of the cases also the joints of the knees are removed. This type of augmentation is primarily the result of the use case, in which it can happen that the feet and partly also the knees are covered by the hood of the vehicle.

4.2.4 Training

33% of the JHMDB-1 training data is used for validation. The focus was not on the tuning of hyperparameters and therefore only varied the batch size, the learning

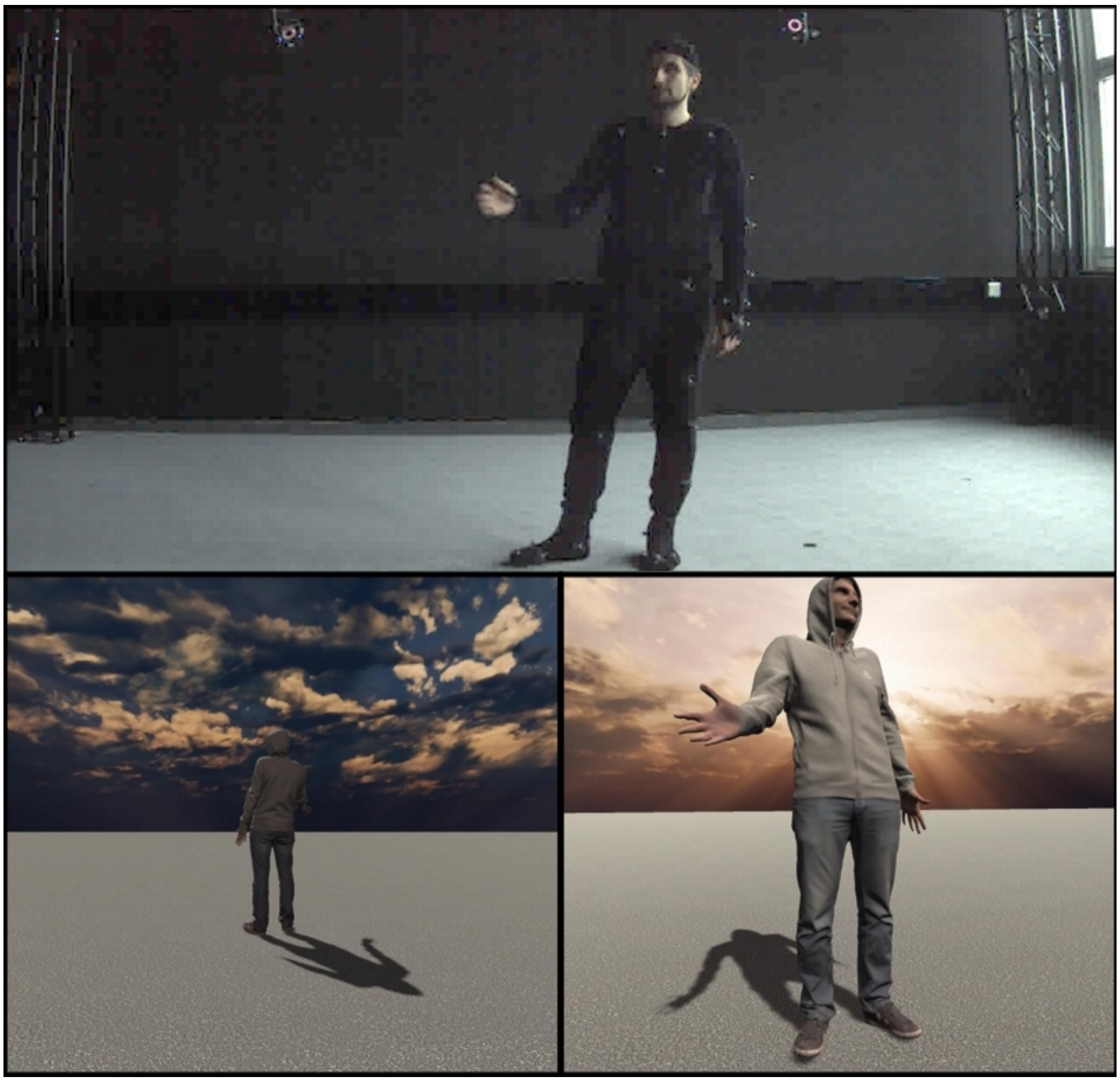


Figure 4.13: Demonstration of virtual sensor information used to train the action recognition algorithm. On top a picture of a real sensor of the motion recording in the motion capture lab and below the simulated scene from two different camera positions.

rate, and the number of epochs to ascertain how to train the network fast and stable. The network is trained with a batch size of 64, an initial learning rate of 0.05, and a standard momentum of 0.9 for 200 epochs (140 on JHMDB-1-GT) for the experiments on the JHMDB dataset. Since the JHMDB dataset is small, a weight decay (L2 regularization [134]) of $5e^{-4}$ is used to counteract overfitting. For optimization, stochastic gradient descent (SGD) and cross-entropy loss are used. The learning rate is reduced every 50 epochs by a factor of ten. The classes in the JHMDB dataset are not evenly distributed, especially as far as the number of EHPIs per video sequence is concerned, as they vary in length. Therefore, a sampling per epoch is applied that outputs a balanced number of samples for each class by reusing samples from classes with few samples and using only a subset of samples from classes with many samples. The same parameters are used for the use case dataset, but since there is considerably more data available, the batch size is adjusted to 128. The network is trained with five different seeds to exclude random effects during weight initialization and ensure the results' reproducibility. Therefore, results are reported as the mean value with standard deviation over these five runs.

4.2.5 Experiments

4.2.5.1 JHMDB Evaluation

The PoTion [76] approach, which was SOTA during the development of EHPI, combines their pose-based action detection with the multi-stream approach I3D [77] and achieves a total performance of 85.5% on the JHMDB dataset. Since only the three actions 'idle', 'walk' and 'wave' are required in the use case, which can also be distinguished purely with pose data, and real-time is a prerequisite, the pure pose-based EHPIs are used in this work. Therefore, the results are compared with parts of other work that also report results for pure pose-based algorithms. The results are summarized in Table 4.3 in terms of accuracies. JHMDB results are reported as the mean value over the three dataset splits. In cases where the pose recognition pipeline was unable to find a human in a video sequence, the action recognition algorithm could not be applied. Thus, that sample was counted as recognized falsely. In 904 of 928 videos, the algorithm was able to recognize a human skeleton in at least two frames and thus created an EHPIs and performed the action detection. For cases where more than one person was detected in a video, the one with the highest pose score is used.

PoTion [76] was outperformed by a margin of 3.5% on the whole JHMDB dataset. On JHMDB-1 Zolfaghari *et al.* [70] provided results too. PoTion was outperformed by a margin of 1.2% and Zolfaghari *et al.* [70] by a margin of 14.8%. Using only the ground truth pose information provided by the JHMDB Dataset, the results of PoTion outperform our results by a margin of 5.3%. This can be either caused

Table 4.3: Mean classification accuracy on the JHMDB dataset compared to other pure pose-based algorithms.

Method	JHMDB	JHMDB-1	JHMDB-1-GT
PoTion [76]	57.0	59.1	**70.8**
Zholfaghari *et al.* [70]	N/A	45.5	56.8
EHPI (ours)	**60.5 ± 0.2**	**60.3 ± 1.3**	65.5 ± 2.8

because our pose recognition pipeline provides superior pose information or because PoTion was applied to a cropped image around the actuator.

4.2.5.2 Automotive Parking Lot Use Case Evaluation

To evaluate the system, two types of results were compared. First, it is reported how many action sequences were correctly recognized, denoted by Accuracy (Seq). Since a sequence can sometimes last several seconds and the total detection consists of the accumulated predictions of the individual EHPIs, it is also useful to indicate how many of the individual EHPIs are correctly detected, denoted by Accuracy (EHPIs). The results are shown in Table 4.4.

Table 4.4: Mean classification accuracy on use case data

Method	Accuracy (Seq)	Accuracy (EHPIs)
SIM (Pose)	80.74 ± 2.77	69.72 ± 1.80
SIM (GT)	79.26 ± 3.78	67.78 ± 1.86
SIM	81.48 ± 3.31	70.64 ± 2.60
Real only	99.26 ± 1.48	95.75 ± 1.65
SIM + Real	99.26 ± 1.48	97.07 ± 1.80

With real data only, 99.26% of the test sequences were classified correctly. The misclassified sequence is an ‘wave’ sequence that has been classified as ‘idle’. The false detection was probably caused by the fact that the waving in this sequence was executed with the left hand, for which only little training data was available. The overall great results are because the use case is rather focused and sufficient similar training data is available. With 81.48% correctly recognized sequences and 70.64% correctly recognized EHPIs when trained purely on simulated data, there appears to be no considerable domain shift between simulated and real training data. It was also found that the performance is slightly better when the noisy pose data from the proposed pose estimation pipeline is used as ground truth rather than using the pose information directly from the motion capture system, hinting that it is beneficial to use both ground truth sources. This is probably because the algorithm is also challenged with noisy pose estimation information on real-world data. As the standard deviation shows, the hyperparameters are not yet optimal for training, but network tuning is not the focus of this work. By combining real and simulated training data for action detection, overall detection rate of all EHPIs was increased by 1.32% to 97.07%. Considering how easily and quickly the simulated data can be generated, using the simulation approach, at least as an addition to real data, is very promising.

4.2.6 Discussion

An efficient pipeline was built to recognize humans in real-time, estimate and track their poses, and recognize their current action. Human poses were encoded over a fixed period into an image-like data structure which can be used in classification CNNs to perform action recognition. The EHPI based action detection delivers SOTA performance compared to other pose-based algorithms and still runs in real-time. In future work, it should be investigated how scene properties and context can be encoded into an EHPI to be able to recognize actions that are not distinguishable on pose data only. In addition, the requirements of the automotive parking lot use case were realized with the presented pipeline. Action recognition results could be transferred to other sensors, environments, and people in first tests. It was shown that using simulation data in combination with real data is suitable for the enrichment of training of action detection algorithms. The results obtained on the purely simulated training data are also auspicious. This approach is further evaluated in section 4.3 to determine if the portion of real data for the training can be reduced further or even be omitted. Since the 2D human pose data is dependent on the camera position, this method is not yet directly applicable on moving platforms. To enable stable detection on moving platforms, one could either try to calculate out the own motions, but this is error-prone. Alternatively, a different approach is investigated in section 4.5 to estimate 3D human pose data in a local pose coordinate system independent of the camera position.

4.3 Action Recognition Using Only Simulated Training Data

The work shown in section 4.2 showed the potential of using 2D human pose data as an abstraction layer to pure visual input to neural networks. This observation is investigated further in the following application of simulated data on the example of action recognition. Section 4.3.2.1 contains a demonstration of how a model can be trained with purely simulated data and then used directly on real data. To achieve this, abstract input data is used for a recurrent neural network in the form of human joints instead of raw sensor data. The effect of using noisy human pose data from an algorithm compared to perfect ground truth pose data to train the network is also investigated. This work further shows how only simulation data can be used to train a simple pose-based action recognition algorithm that can be applied to real data without the application of domain transfer methods.

The main contributions in this part of the work are:

1. It is shown how to produce raw data for the simulation framework and how to use this framework to simulate camera sensors that generate data with the corresponding ground truth for the two scenarios demonstrated in this work.

2. It is shown if and how a human pose estimation algorithm can be fine-tuned using only simulated data based on motion capture data and 3D avatars to recognize corner case poses, which it could not detect when trained only on standard datasets.

3. With data produced in a laboratory environment, it is demonstrated that the training of a human pose-based action recognition algorithm is possible on synthetic data only, which can be directly applied to real sensor data.

4.3.1 Simulated Data

A 3D scanned model of the person performing the motion recordings in the motion capture lab has been used for the action recognition experiment. The movements in this work vary from ‘idle’, ‘walk’, ‘wave’ and ‘sit’ to ‘jump’. The environment is not very relevant in the experiments in this work because the action recognition algorithm is purely based on pose data. Thus, it is only secondarily dependent, through the pose recognition, on visual features, if at all. Thus, a static background consisting of a flat area surrounded by a simple skybox (see Figure 4.14) is used.

4.3.2 Virtual Content Generation

Figure 4.15 shows some examples of automatically generated ground truth data. Virtual camera sensors were used to record the use cases’ data with annotated 2D

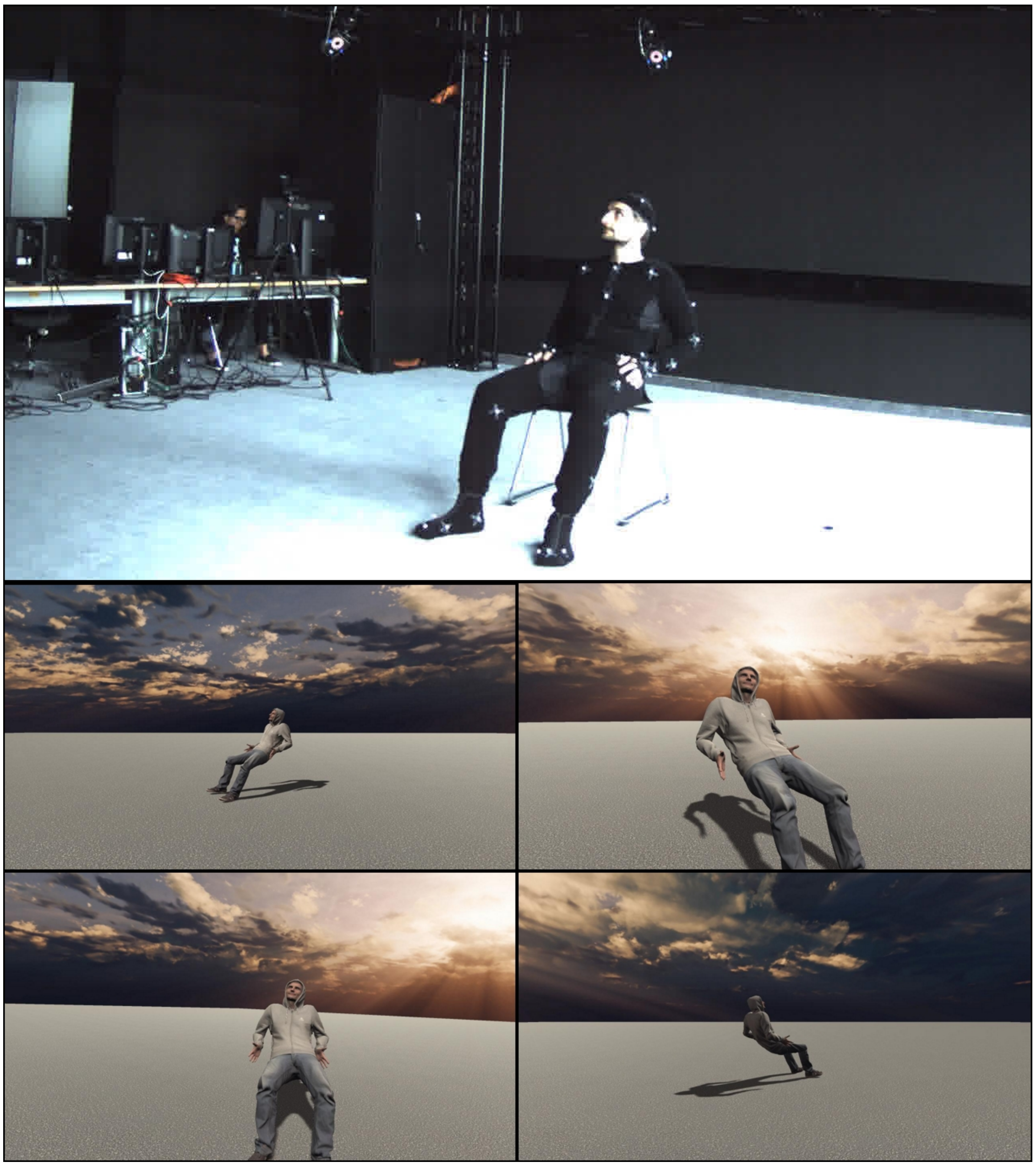


Figure 4.14: Demonstration of virtual sensor information observing a sitting human used to train the pose-based action recognition algorithm. Note that no actual chair was simulated because only the pose is relevant. The real camera sensors were positioned such that there is no occlusion from the real chair.

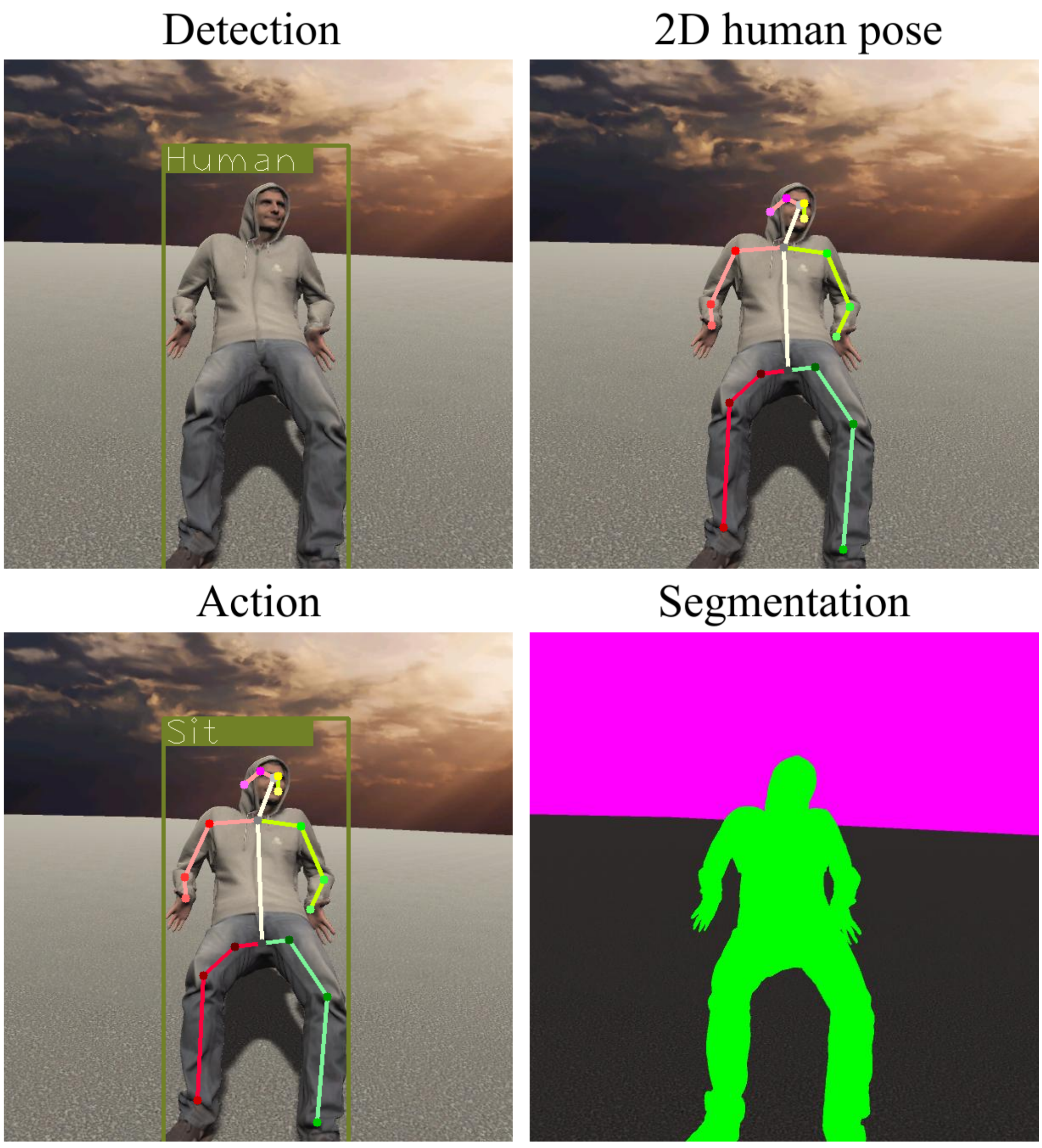


Figure 4.15: Automatically generated ground truth data for a frame recorded by a virtual camera sensor.

human joints as training ground truth for the corner cases and an action label for each frame for the action recognition experiment.

4.3.2.1 Experiments

In the following, two laboratory experiments are presented to investigate whether a recurrent neural network for action recognition can be trained with purely synthetic data and how it performs on real data. To achieve this, RGB cameras captured real test data during the recording of a participant in the motion capture laboratory. The real lab setup was reproduced accurately in the simulation by using virtual cameras positioned in a similar position as the real cameras. 3D scans of the person recorded in the motion capture lab were used, yet with different clothing, to rule out transfer effects between different body proportions. A purely pose-based algorithm for action recognition was applied. As such, temporal action recognition will not be confounded by noisy visual features.

4.3.3 Datasets

4.3.3.1 ActionSim

In the motion capture lab, one person performed the five actions ‘idle’, ‘run’, ‘wave’, ‘jump’ and ‘sit’ continuously ten times for about 11.5 seconds each. These actions were selected for this experiment because for a first demonstration of the transfer from a model, which was trained on purely simulated data, to real data, well differentiable actions should be used. Every motion capture sequence contains only motions that fall into one action class. Transitions between different motions are not recorded. This allows the motion sequences to be used in the simulation without manual post-processing. The person was recorded by two real camera sensors, which were placed directly in front and on the participant’s right side. The virtual camera arrangement is visualized in Figure 4.16.

The dataset consists of 100 sequences, recorded with a resolution of 1280x720 at 30 FPS, resulting in around 17,800 frames per camera. Using this data three training sets were created, ‘ActionSim (GT)’, ‘ActionSim (Pose)’ and ‘ActionSim (Both)’. ActionSim (GT) and ActionSim (Pose) contain the joint positions for the human for each frame of each of the six virtual camera sensors. The datasets differ in the source of the joint positions. In ActionSim (GT), the human joints are from the simulation ground truth data produced by the simulation framework. The ActionSim (Pose) dataset was created because the pose-based action recognition algorithm has to work with joints from pose recognition algorithms that may be noisy and have problems with false detections or no detection. In this dataset, the pose recognition algorithm, demonstrated in section 4.3.4, was applied to the virtual camera images to obtain correspondingly noisy data. Due to the similar issues on

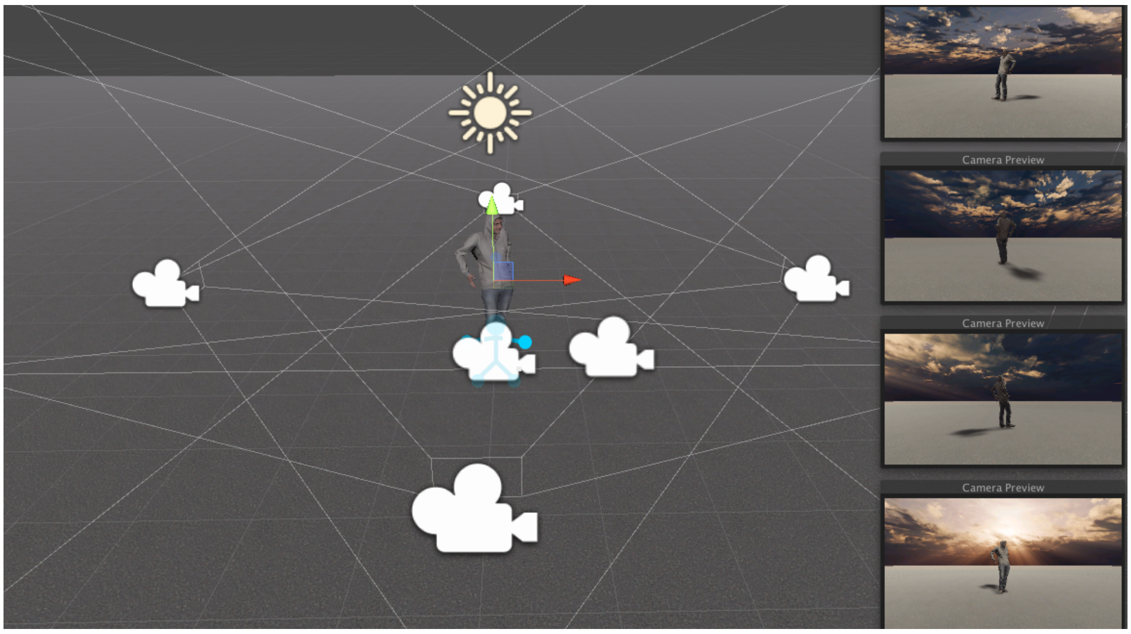


Figure 4.16: Six virtual camera sensors positioned around an animated human. Every camera is oriented towards the human. Two of them are arranged similarly to the position of the real cameras, the other four view the scene from orthogonal directions, from front, rear, left, and right.

simulated data as on real data, like missing and false recognized joints as visualized in Figure 4.4, it is expected, that the pose recognition algorithm delivers noisy outputs, similar to the output on real-world data, by applying it to the simulated camera images. The dataset ‘ActionSim (Both)’ consists of the entire ActionSim training dataset, thus using action pose ground truth from both ground truth data and the pose recognition algorithm.

The test part of the ActionSim dataset consists of the 100 sequences recorded by the two real camera sensors. Each sequence was assigned one of the five action classes used as a label.

4.3.3.2 SimTransfer Dataset

An additional experiment is presented to test the transferability to scenarios apart from the laboratory scenario. For this experiment, a dataset, called ‘SimTransfer dataset’, was created in which the same person who performed the actions in the motion capture lab was recorded performing five sequences per action in an office environment. To record the sequences, a standard webcam was used that was positioned differently from the lab cameras. Each sequence is ten seconds long. Again, a resolution of 1280x720 at 30 FPS was used. This dataset was used purely for testing, and no parts of it were used as training or validation data.

4.3.4 Pose Recognition Algorithm

To retrieve the pose data required for the action recognition algorithm, the pose recognition pipeline described in section 4.2 was used.

4.3.5 Action Recognition Algorithm

In contrast to the previous experiments with the new EHPI action recognition approach, this work applies a standard approach based on a recurrent neural network for pose-based action recognition, which is very fast to train. This is because only the applicability of simulated data in a defined laboratory scenario is to be tested and, accordingly, rapid trainability has higher priority than absolute recognition performance. The network consists of a two-layer LSTM, with an input dimension of $m \times n$. Factor m denotes the time window (TW) length that is set to $m = 32$ frames. This means that the system looks at just over a second of motion to detect an action using a 30 FPS camera sensor. Factor n denotes the number of two-dimensional joint coordinates. 15 joints of a human skeleton are used in this work. Thus, n is set to $n = 30$. The LSTM hidden layers have a dimension of 64 and are followed by a fully connected layer, which classifies five action classes. The network was trained with the stochastic gradient descent (SGD) optimizer and use

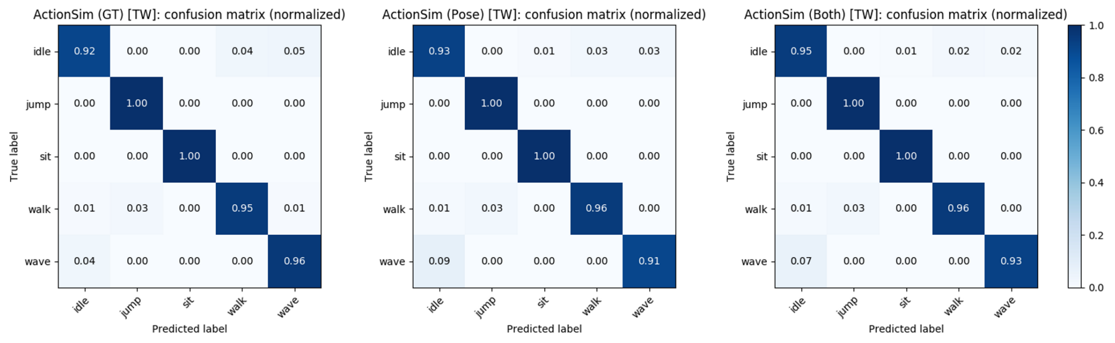


Figure 4.17: Confusion matrices for the time window results from ActionSim (GT), ActionSim (Pose) and ActionSim (Both). It can be seen that the errors in GT and Pose are different, and the combination of both pose sources provides the best overall result.

the cross-entropy loss. A learning rate of 0.01, a momentum of 0.9, and a batch size of 256 for 200 epochs were applied.

4.3.6 Data Preprocessing

For data normalization, first human joints outside the actual image area are removed, which is especially important for ground truth data. Then the 2D joint coordinate pairs are normalized over the time window used in the recurrent neural network. This normalization takes place independently for the horizontal and vertical positions by setting the respective minimum value over the full-time window to zero and the maximum value to one. The remaining values are scaled accordingly. To obtain higher variability, the training data was augmented by flipping the joint positions horizontally in 50% of the samples. In 50% of these flipped cases, the right and left joints' identifier was exchanged to vary the orientation of the human to the camera.

4.3.7 Results

The results are reported as accuracy (sequences) and accuracy (TW) of an action. Accuracy (sequences) refers to the number of correctly classified video sequences. The most frequently classified class during the sequence is used as the sequence action label. Since the sequences are relatively long, the recognition accuracy of each particular time window are reported in accuracy (TW). To counteract the possible effects of the initial weightings, all tests were carried out five times with different random seeds and report the results as mean values with standard deviation.

4.3.7.1 ActionSim Dataset

Table 4.5: LSTM based action recognition baseline: Trained purely with synthetic data, evaluated on related real sensor data from the lab dataset.

Training Dataset	Accuracy (Sequences)	Accuracy (TW)
ActionSim (GT)	98.6 ± 0.49	96.63 ± 0.59
ActionSim (Pose)	97.8 ± 1.16	95.93 ± 1.37
ActionSim (Both)	**98.8 ± 0.40**	**96.81 ± 0.80**

All results for the ActionSim dataset are listed in Table 4.5. The data reveals that the transferability of the model, which was only trained with simulated data, to real data is provided. No domain shift effect seems to influence the result. Not only were the sequences correctly classified in 98.8% of the cases, but 96.81% of the particular time windows were also correctly classified. As the experiment shows, it is beneficial to use ground truth pose information from motion capture ground truth data in combination with data from the pose recognition algorithm. The training data created by the pose recognition algorithm gives slightly worse results, possibly because there is a bias in it, which the real sensor data did not record. A reason might be that four of the six virtual camera sensors placed around the person are further away than the two real camera sensors, which challenges pose estimation and further action recognition.

As can be observed in the confusion matrices in Figure 4.17, false recognition in ActionSim (Pose) and ActionSim (GT) differs, which explains why the combination of both leads to better results. In total, six of the 100 sequences were confused. However, these differed across random seeds, indicating that the training was not yet optimal and by adjusting the training parameters, even though erroneous sequences could then be correctly classified. Most of these cases were confusions between the actions ‘idle’, ‘wave’ and ‘walk’. Figure 4.18 shows four frames from an ‘idle’ sequence that was recognized as ‘walk’. As can be observed, in this case, the pose recognition was often incorrect due to occlusion, which caused confusion between the left and right legs. This is similar to the pose information that occurs when walking and probably caused incorrect action recognition. Furthermore, with ‘idle’ sequences that were recognized as ‘waving’, incorrect pose information due to occlusion seems to be the main problem, as strong movements of the occluded arm occur from frame to frame.

One of the ‘idle’ sequences was falsely classified as ‘sitting’. Although this is surprising at first, the probable reason becomes clear in Figure 4.19. If only the normalized pose information is considered, it is very similar in the frontal view of a ‘sitting’ and an ‘idling’ person. Since not much movement was involved in the two sequences, the network has to work mainly with spatial information. However, in

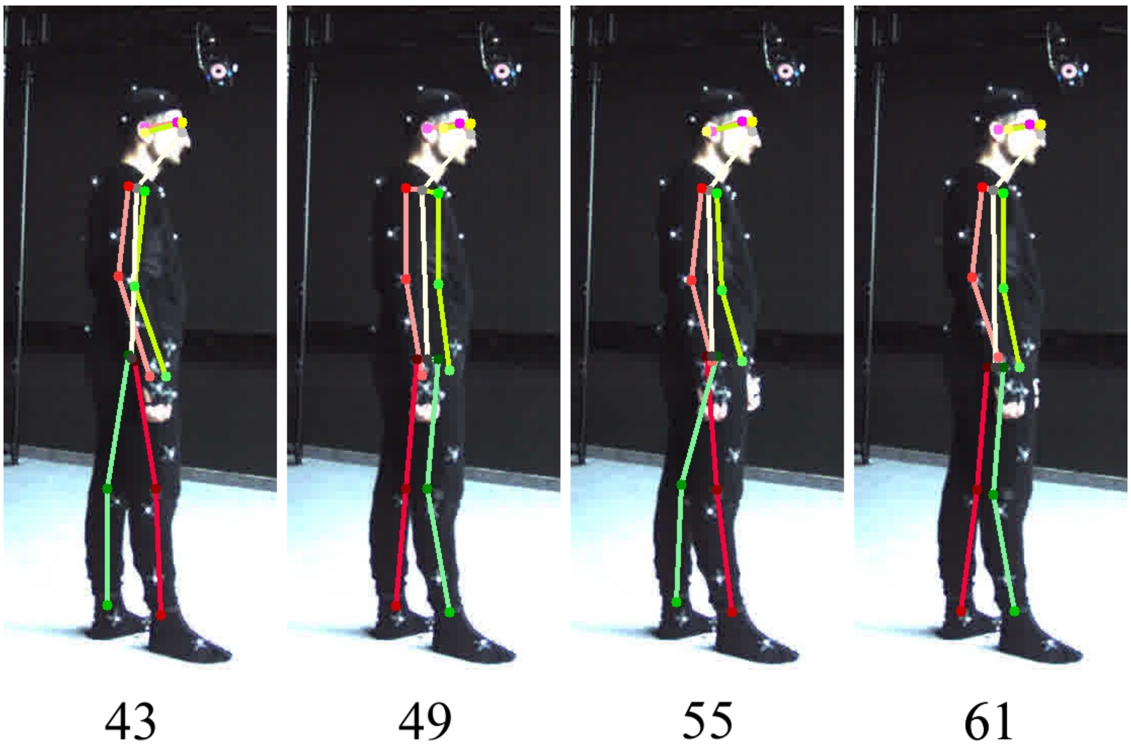


Figure 4.18: Four frames from an idle scene that was classified as walking.

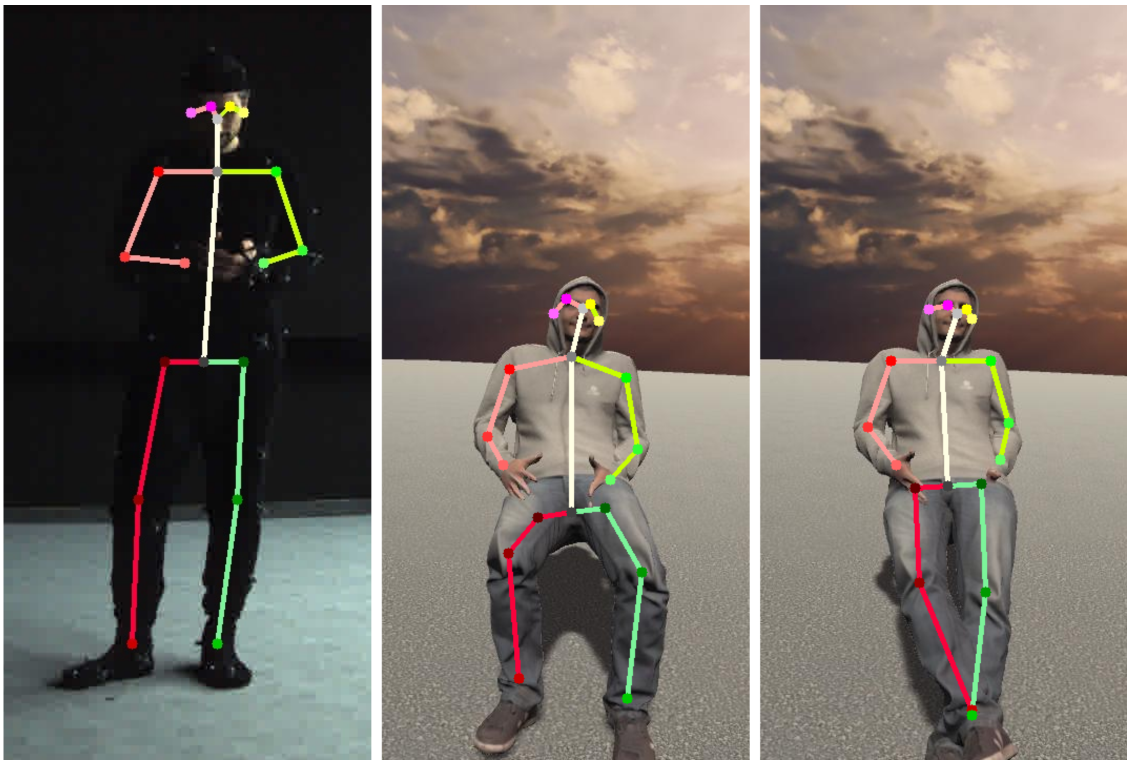


Figure 4.19: Exemplary representation of a frontal view in which the skeletons resemble a sitting and an idling person.

most cases, the network was still able to distinguish between the two actions. The confusion only occurred with one random seed.

4.3.7.2 SimTransfer Dataset

A good transfer of the results to real data was expected because the situations were deliberately kept very similar, but the excellent results exceeded this expectation. Therefore, the SimTransfer dataset (see section 4.3.3.2) was created to evaluate on an independent dataset in another scenario. The recorded person is the same as in the laboratory experiment, so the style of executed actions is similar.

Table 4.6: Simple LSTM based action recognition baseline results: Trained purely on synthetic data, evaluated on unrelated real sensor data from the SimTransfer dataset.

Trainingset	Accuracy (Sequences)	Accuracy (TW)
ActionSim (GT)	**100**	96.74 ± 0.94
ActionSim (Pose)	99.2 ± 1.6	98.29 ± 0.95
ActionSim (Both)	**100**	$\mathbf{98.52 \pm 0.38}$

The results of this experiment are listed in Table 4.6. All sequences could be classified correctly. The classification of the particular time windows was even better than in the laboratory experiment itself. This may be explained by the fact that no motion capture suit was worn, and the lighting situation was better, which made detection easier for the pose recognition algorithm, thereby providing cleaner input data for the action recognition algorithm. These results show a very promising tendency towards employing purely synthetic data production to train data-driven algorithms. Although, only a straightforward LSTM was used for the action recognition, it performed surprisingly well on pose-estimation results based on webcam sensor data. Based on the time window recognition rate, the inclusion of noisy data from the pose algorithm also offers an advantage over the perfect ground truth data. This demonstrates that it is useful to generate visual, simulated sensor information and apply the pose recognition algorithm to this simulated sensor data to obtain more realistic ground truth data for pose-based action recognition. The results on the SimTransfer dataset were corroborated in a live test in which the same person performed actions similar to those in the dataset. Only transitions between different actions lead to misclassification. An example is a transition from sitting to walking, during which the jump action is detected. This is caused by a strong vertical movement of some joints, which occurs in the training data only during the jump action. However, this is expected since transitions between actions are not part of the training data. If the action 'get up' was included in the data-set, the model should distinguish between jumping and getting up by the spatial

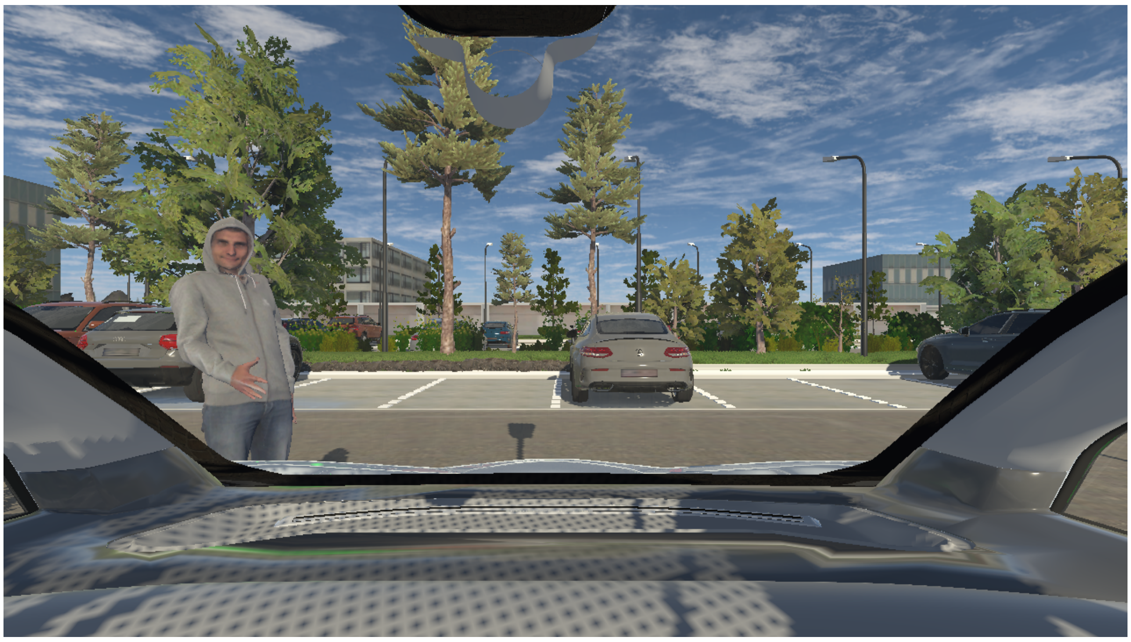


Figure 4.20: Example of generated simulation data to train an autonomous car people's intent to let them drive out.

arrangement of the joints. Action transitions would be a worthy focus of future research.

Since the simulation environment is easily adaptable, the approaches shown in this work can also be transferred to other simulation environments for specific problems. For example, the simulation can be used to generate data for the training of autonomous vehicles as in Figure 4.20, which shows simulated sensor data for a person waving a car out.

4.3.8 Discussion

Based on action recognition, it was shown that by data abstraction through intermediate human pose-estimation, algorithms trained with synthetic data can be applied to real data without domain transfer methods. An almost perfect performance of such an algorithm on real data in the laboratory experiment was achieved. Through an additional experiment with real data, first evidence that the results are transferable to real scenarios was provided. It was beneficial for the pose-based action recognition algorithm to generate ground truth data by applying a pose recognition algorithm to simulation data in combination with using perfect pose information from the motion capture system. The experiments demonstrated, that the use of human simulations provides sufficient data variety to improve the quantitative and qualitative constitution of the overall dataset. The experiments showed great potential to use simulation to train data-driven algorithms to understand human pose and actions. Therefore, the usage of simulated training data seems to be a promising and valid approach to further advance machine understanding of human behavior, especially for encounters of autonomous intelligent transportation systems with pedestrians. One shortcoming in 2D human pose estimation is that some poses are indistinguishable in 2D, as such the following experiment will extend the 2D approach to 3D, which will also be beneficial for further tasks, like human body and head orientation estimation.

4.4 PedRec: Multi-Task Pedestrian Recognition

In addition to the experiments shown on 2D human pose data in section 4.2.6, 3D human pose data is valuable information for action recognition, as some poses might be indistinguishable in 2D (see for example Figure 4.19 in the previous section). To solve such problems, the following section shows a system that provides 3D human pose data in addition to 2D human pose data. The system shall be the basis for further experiments. Therefore, the focus is on a straightforward structure, a simple and stable training process, high performance on consumer hardware for live experiments, and extensibility. Furthermore, the system should be independent of specific camera configurations. Thus, the architecture should not include any assumptions about the sensor hardware. The focus is on single frame estimation, since the temporal component for action recognition is already contained in the EHPI action recognition approach. In addition to 3D human pose data, orientation information about a person's body and head is also relevant for human recognition systems. Especially in pedestrian recognition, this information can be valuable to perform path planning or to detect if a pedestrian notices a vehicle or not. Therefore, body and head orientation estimation are added to the network as an additional task. All tasks in the PedRecNet are related and based on the same input data, so all tasks should be implemented in the same network using a multitask approach. Since there are only a few real datasets available for 3D human pose recognition and especially for body and head orientation estimation, simulation data was used to improve these parts of the PedRecNet and to enable training in the first place. The skeleton-based action recognition results presented in sections 4.2 and 4.3 support the assumption that using abstract pose information rather than just visual information enables the transfer of simulated training data to real data. This work hypothesis is corroborated in more detail in experiments using simulated training data. The developed network, the datasets used, and evaluate 3D human pose recognition as well as the body orientation estimation on several real and simulated datasets are described in the following.

The entire system and novel simulation data has been made public under the MIT license[4].

4.4.1 Definitions

3D skeletons 3D skeletons are represented by the PedRec skeleton hierarchy (see Figure 3.13) in a left-handed Cartesian y-up coordinate system which is pelvis-centered (see Figure 4.21). The skeleton is normalized to a $3 \times 3 \times 3$ meters volume. Other approaches (for example [39]) use a volume of $2 \times 2 \times 2$ meters, but the volume was extended in this work to be able to correctly map

[4]`https://github.com/noboevbo/PedRec` (accessed on 2022-03-06)

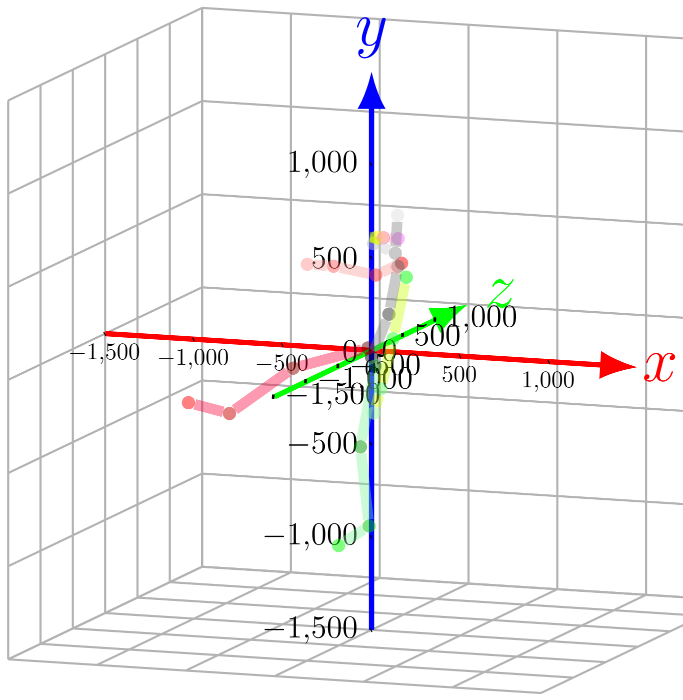


Figure 4.21: Local skeleton 3D coordinate system centered at the pelvis of the skeleton.

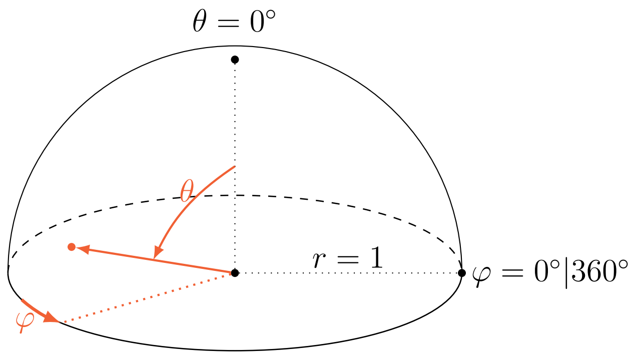


Figure 4.22: Visualization of the orientation estimation for each, the body and the head. The orange dot shows an example point on a 3D sphere, visualizing an orientation. The standard notation from ISO 80000-2:2019 [135] is used for spherical coordinates. As only the polar angle θ and azimuthal angle φ are required, a unit sphere and with $r = 1$ was used.

fully stretched, tall people.

Orientations Orientations are represented in a spherical coordinate system following the ISO 80000-2:2019 (Item No. 2-17.3) standard [135], which states that spherical coordinates should be written as (r, θ, φ) where r is the radial distance, θ the polar angle and φ the azimuthal angle (see Figure 4.22). As the orientation estimation only requires the angles and a unit sphere with $r = 1$ is used, it is only required to predict θ and φ.

4.4.2 Network Architecture

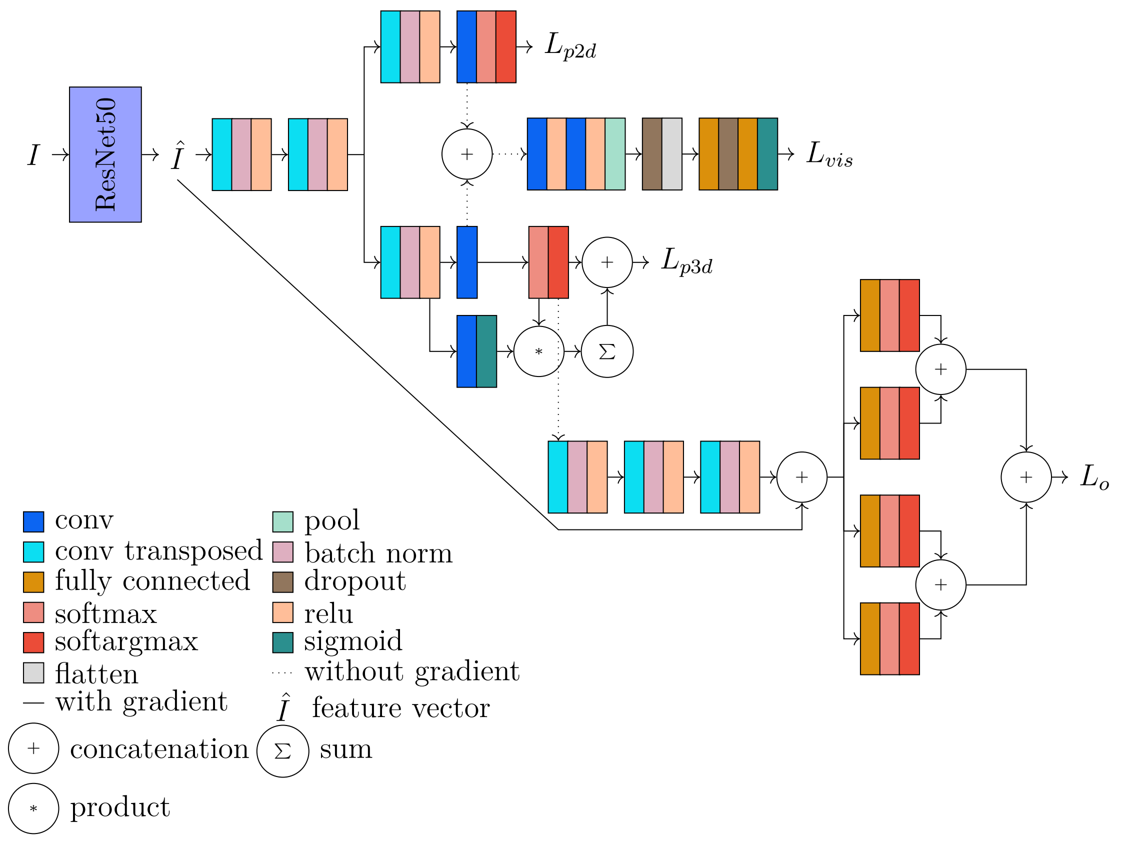


Figure 4.23: Simplified PedRecNet architecture. The input to the PedRecNet is an RGB cropped bounding box image of a human. In this work, an input size of 192×256 is used. A ResNet50 is utilized as the feature extractor. The dotted connection lines indicate connections which are used only in the forward pass, but do not allow gradient flow in the backward pass.

The overall network architecture is shown in Figure 4.23. The PedRecNet expands the 2D human pose estimation approach proposed by Xiao *et al.* [30]. The PedRecNet architecture is based on a ResNet50 backbone for feature extraction, but other backbones could be used as well. The ResNet50 architecture was chosen as a compromise between accuracy and performance. The inputs I are always images cropped to the size of certain bounding boxes. Features $\hat{I}$ are extracted using the feature extraction part of the network. The following network heads are used to fulfil the various tasks:

2D Human Pose Estimation The 2D human pose estimation part is based on three transpose convolution blocks with which joint heatmaps are generated from the extracted features (see section 4.2.1.2).

3D Human Pose Estimation In PedRecNet, the 2D human pose estimation architecture was extended to include 3D human pose estimation. For this purpose, two transpose convolution blocks are used as a common basis and then split into a 2D and a 3D path. These have basically the same structure. The output in the 2D path corresponds to 2D image coordinates. The 3D path, leading to L_{p3d}, corresponds to the estimation of the x and y coordinates of a joint relative to the pelvis and additional depth estimation of the z coordinate using a sigmoid map. Another change from the previous 2D human pose estimation approach is the post-processing of the heatmaps. In the approach shown in section 4.2, the heatmaps were output from the network, and a Non-maximum suppression (NMS) was used to determine the coordinates. This has the disadvantage that the subpixel coordinate values are not available in the network. The NMS approach also implies that artificial heatmaps have to be generated to be used as training data labels. The PedRecNet applies a softargmax layer, which determines the coordinates from the heatmaps in the network inside the network. This process is described in more detail in section 4.4.3. The softargmax layer allows using estimated joint coordinates as inputs into other parts of the network. For example, the 3D joint positions are used as input into the orientation estimation part of the network.

Head & Body Orientation Estimation The head and body orientation part of the network leads to L_o and uses the visual features $\hat{I}$ as well as 3D coordinates from the 3D human pose estimation part of the network. The 3D joint position coordinates are scaled up into a feature space via 1D transpose convolution blocks. These are concatenated with the visual features $\hat{I}$ such that the orientation estimation can use direct information from the image in addition to the noisy and potentially erroneous, 3D human pose. This concatenated feature vector is input into a fully connected layer which generates a one-dimensional heatmap for each, the polar angle θ and azimuthal angle φ (see Figure 4.22) of the body as well as the head. From these one-dimensional orientation heatmaps, the corresponding normalized angle is extracted using 1D softargmax.

Human Joint Visibility To classify the visibility of labels, a standard classification head was added to the network leading to the path to L_{vis}.

The PedRecNet outputs 2D human joint positions as pixel coordinates in the bounding box's coordinate system, normalized between zero and one. The 3D human joint positions are output as 3D coordinates relative to the pelvis position, which are normalized between zero and one. Orientations are output as angles between $0 - 180°$ for θ and $0 - 360°$ for φ, and are also normalized between zero and one.

The used loss functions are described in section 4.4.5.

4.4.3 Heatmap-based Workflow

As shown in the network overview, most task-specific paths in the network produce outputs based on heatmaps. The general pipeline of such an approach is shown in Figure 4.24.

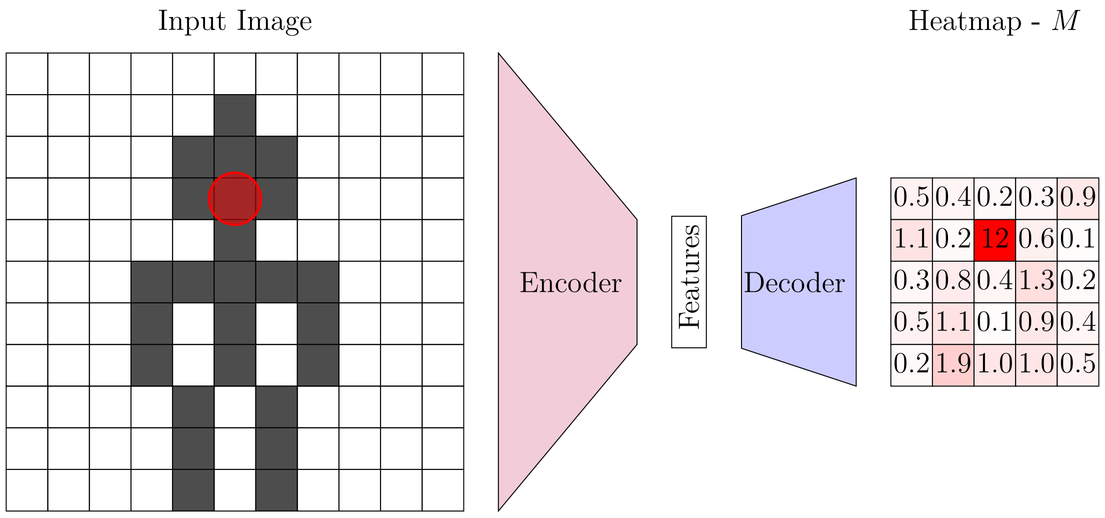


Figure 4.24: Heatmap-based Prediction Pipeline: Encoder - Decoder

Features are extracted from an input image via an encoder. A decoder produces a heatmap from these features, representing geometric information such as a joint position. Classically, such a heatmap is output by the network and in post-processing via a NMS the deflections on the heatmap above a certain threshold are output (see, for example, [6], [27] or [30]). To train such a network, artificial heatmaps are generated with deflections in the form of a noise distribution as ground truth. In PedRecNet, the joint coordinates should be available for other network parts. For example, the orientation estimation requires 3D joint coordinates as input. Since a NMS cannot be performed efficiently within a network, an alternative would be an argmax layer. However, the argmax function cannot be used directly, as it is not differentiable, and thus, the gradient cannot be calculated. One other problem with the argmax would be the support of fractional positions. As heatmaps are usually smaller than the input, the argmax would lead to imprecise results, depending on the scale difference of the heatmap and the desired output. An alternative is the softargmax function. The softargmax for a K-dimensional vector z uses the softmax, which is defined as:

$$\text{softmax}(z)_j = \sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \dots, K \tag{4.9}$$

The softmax $\sigma(z)_j$ can be derived with respect to z_j as follows:

$$\begin{aligned}
\frac{\partial \sigma(z)_j}{\partial z_j} &= \frac{e^{z_j} \sum_{k=1}^{K} e^{z_k} - e^{z_j} e^{z_j}}{(\sum_{k=1}^{K} e^{z_k})^2} && (4.10)\\
&= \frac{e^{z_j} (\sum_{k=1}^{K} e^{z_k} - e^{z_j})}{(\sum_{k=1}^{K} e^{z_k})^2} && (4.11)\\
&= \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \frac{\sum_{k=1}^{K} e^{z_k} - e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} && (4.12)\\
&= \sigma(z)_j (1 - \sigma(z_k)) && (4.13)
\end{aligned}$$

And with respect to $z_k \neq z_j$ as follows:

$$\begin{aligned}
\frac{\partial \sigma(z)_j}{\partial z_k \neq z_j} &= \frac{e^{z_j} e^{z_k}}{(\sum_{k=1}^{K} e^{z_k})^2} && (4.14)\\
&= -\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \frac{e^{z_k}}{\sum_{k=1}^{K} e^{z_k}} && (4.15)\\
&= -\sigma(z)_j \sigma(z_k) && (4.16)
\end{aligned}$$

As such, the gradient of the softmax function can be derived. The softargmax is defined as:

$$\text{softargmax}(z) = \sum_{j=1}^{K} \sigma(z)_j j \tag{4.17}$$

Thus, the softargmax function is usable in the backpropagation algorithm. The softargmax function has been applied in joint-based approaches before, for example for face landmark detection [136] and human pose estimation [137, 45].

In contrast to NMS this approach can exclusively be used for heatmaps that contain only one position per heatmap (for example, a joint position in the form of x and y coordinates), which is given in PedRecNet. To visualize the behavior of the softargmax function, it can be split into the softmax part and the index multiplication. As such, first, the softmax is calculated for every element m_{rc} in a heatmap M with size $R \times C$. The 2D softargmax function is defined as:

$$\text{softargmax}_{x,y}(M) = s_{x,y}(M) = (\sum_{r=1}^{R}\sum_{c=1}^{C}\frac{r}{R}\sigma(M_{r,c}), \sum_{r=1}^{R}\sum_{c=1}^{C}\frac{c}{C}\sigma(M_{r,c})) \tag{4.18}$$

To further highlight the principles and one downside of this approach, one can visualize the processing steps of the heatmaps. The first step is the application of the softmax, as shown in Figure 4.25.

0.5	0.4	0.2	0.3	0.9
1.1	0.2	12	0.6	0.1
0.3	0.8	0.4	1.3	0.2
0.5	1.1	0.1	0.9	0.4
0.2	1.9	1.0	1.0	0.5

(a) Heatmap M

0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

(b) Heatmap after $\sigma(M)$

Figure 4.25: Softmax visualization

This form of a heatmap with a single maximum value of one is an optimal case. The coordinate extraction step from such a heatmap is shown in Figure 4.26a.

$$\sigma(M_a) = \begin{array}{|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1.0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

(a) $s_{x,y}(M_a) = (0.25, 0.5)$

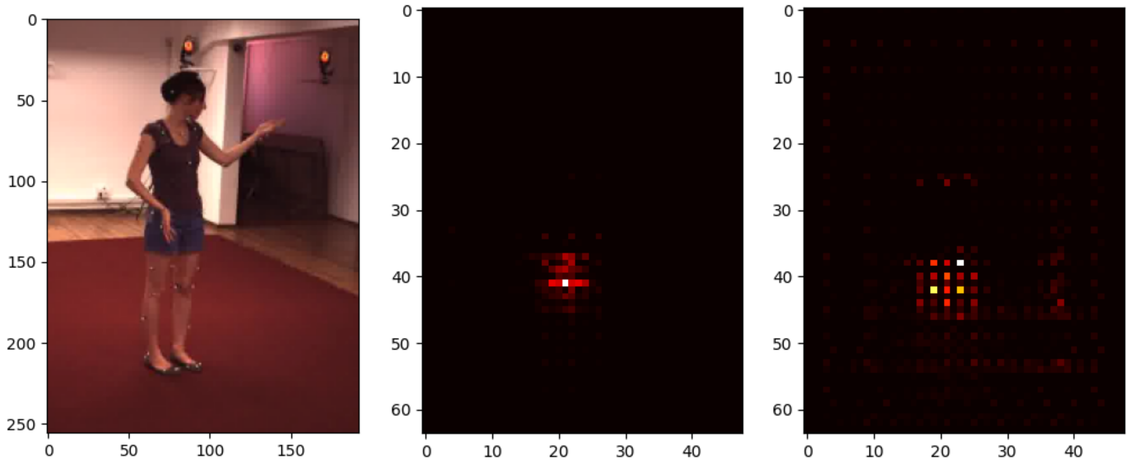
$$\sigma(M_b) = \begin{array}{|c|c|c|c|c|}\hline 0 & .275 & 0 & .275 & 0 \\ \hline 0 & .1 & 0 & .1 & 0 \\ \hline 0 & .05 & 0 & .05 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & .075 & 0 & .075 & 0 \\ \hline \end{array}$$

(b) $s_{x,y}(M_b) = (0.25, 0.5)$

Figure 4.26: Comparison of the same softargmax outputs from different softmax heatmaps.

Luvizon *et al.* use the maximum value of the softmax output as the joint confidence [45]. However, it is important to note that during the PedRecNet training process, the network has not been subject to any other constraints, such as an enforced peakiness of the heatmap. This allows the network to learn alternative representations. An example of this is shown in Figure 4.26b.

If the two softmax heatmaps in Figure 4.26 are compared, it can be seen that different heatmaps can lead to the same output. Furthermore, it becomes clear that the numerical value of the maximum deflection of the softmax output cannot be an indicator for confidence. Examples of an unconstrained softmax output are shown in Figure 4.27.



(a) Input (b) Pose 2D (c) Pose 3D

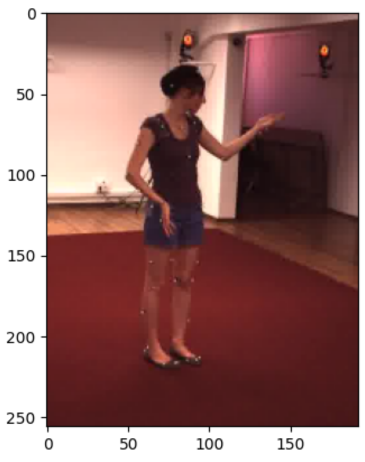
Figure 4.27: Human Pose: Softmax Heatmaps learned without peakiness constraint. Max Val 2D: 0.00011, Max 3D: 0.0015

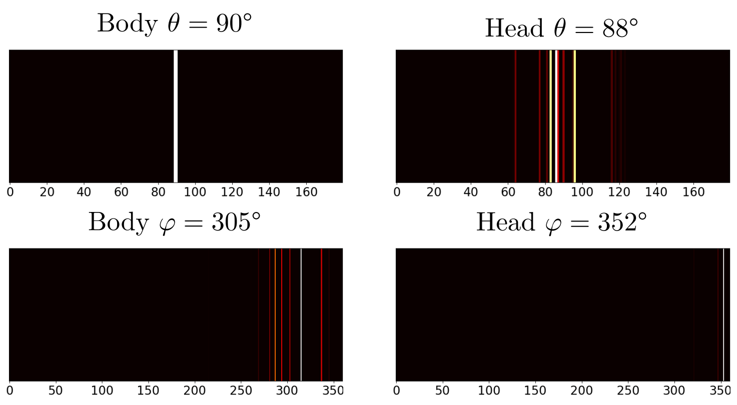
The Pose2D softmax heatmap is rather peaky at the actual location of the joint. In contrast, the Pose3D heatmap contains some more peaks. The network seems to have learned various numerical help constructs, which result in the desired output without a single peak at the actual joint position. One-dimensional heatmaps show similar behavior, as visualized in Figure 4.28.

The head orientation prediction of φ shows the actual prediction of 352°. The body orientation prediction shows multiple peaks for the prediction of φ. The prediction of the heads' θ shows even more peaks, similar to the example of 3D human pose estimation.

Luvizon *et al.* [45] added a constraint in the form of a BCE loss applied on the maximum value of the softmax heatmap with ground truth provided from the visibility label of a joint to enforce the peakiness of a heatmap. Figure 4.29 shows an example output for the PedRecNet trained on Pose2D data only with the BCE loss on the maximum value of the softmax map.

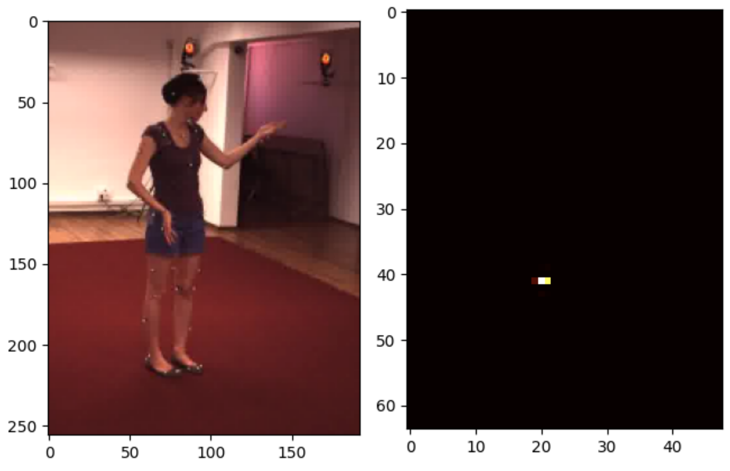
The max value was used after average pooling with a 3×3 kernel and a stride of 1×1, as the heatmap usually takes a Gaussian form to enable fractional coordinate output. The network can apply the peakiness in this case and reaches a similar performance as without this constraint when trained on COCO [25]. Nevertheless, using multiple datasets resulted in a more unstable training which got even more





(a) Input (b) Orientation Softmax Maps

Figure 4.28: Human Body and Head Orientation: Softmax Heatmaps learned without peakiness constraint.



(a) Input (b) Body

Figure 4.29: Human Pose: Softmax Heatmaps learned with peakiness constraint in form of an applied BCE loss on the softmax maximum value based on the visibility label of a joint.

instable when adding additional tasks, like the 3D human pose estimation. This may be caused by the presented PedRec architecture, in which the 2D and 3D human pose estimation share the first two blocks of transpose convolution yet work in different coordinate spaces. The peakiness constraint seems to limit the network in this regard. The initial intention of the peakiness is to use the maximum value of the softmax heatmap. However, as the BCE loss uses the visibility label to enforce peakiness or flatness of the softmax output, it is representing a joint visibility classification task embedded in the coordinate regression task. Would one add a dataset with valid joint coordinates but visibility labels of zero, the network could learn to regress the joint coordinates well by using a mostly uniform heatmap output similar to Figure 4.27c. As such, PedRecNet does not apply additional peakiness constraints on the softmax output, enabling the network to choose its own representation of the heatmap and thus, enable a more stable training with this specific network architecture. Instead, a standard classification head was added to the network, trained to classify the visibility of labels. In the experiments, the classification rate did not differ significantly by using the separate classification head instead of the max value of the softmax output enforced by the peakiness constraint. It could be investigated to model the uncertainty in the network to output the actual confidence in the coordinate regression task. One possibility would be using the loss of the coordinate regression task as a label for the uncertainty task. However, this is beyond the scope of this work and thus, not further investigated.

To summarize, the term heatmap and softargmax are not quite accurate without a peakiness constraint. In an implementation without such a constraint, the heatmaps are rather auxiliary constructs and the softargmax is not a real argmax either. Nevertheless, the term heatmap will be used in the following because the output still corresponds to a position on a heatmap, even if the network does not necessarily operate with peaks internally.

Orientation Estimation Heatmaps In addition to the presented use of a one-dimensional heatmap for orientation estimation, it is also possible to use a two-dimensional heatmap, which allows the orientation estimation to be set up in the same way as the 2D and 3D human pose estimation. For this purpose, the polar coordinates can be converted into Cartesian coordinates (see Figure 4.30).

The figure shows that the ground truth coordinates lie on a unit circle for this purpose. This method was also tested, but the performance was comparable with the one-dimensional approach. However, the one-dimensional approach requires significantly fewer parameters, so the one-dimensional variant was preferred due to the better runtime behavior.

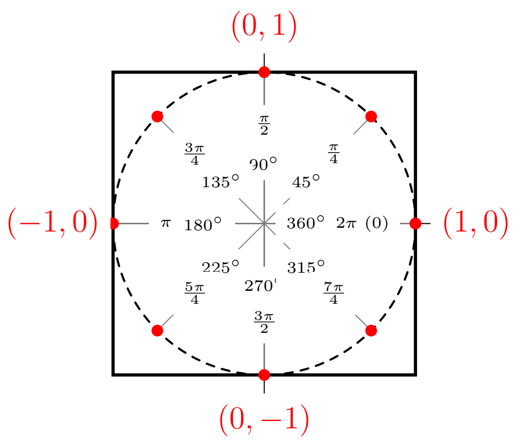


Figure 4.30: Example of how polar coordinates could be represented as Cartesian coordinates on a 2D heatmap.

4.4.4 Datasets

The training and validation datasets of the PedRec network are composed of the COCO [25], the H36M [48], and self-generated simulated datasets. The datasets support different labels, especially orientation estimation labels are only available in the simulated data (cf. Table 4.7).

Dataset	Pose2D	Pose3D	Orientation	Action
COCO [25]	✓	X	O	X
H36M [48]	✓	✓	X	X
TUD [67]	X	X	O	X
SIM-ROM	✓	✓	✓	X
SIM-Circle	✓	✓	✓	X
SIM-C01	✓	✓	✓	✓

Table 4.7: Overview of the used datasets and the supported labels. COCO [25] (MEBOW [65]) and TUD [67] orientation annotations provide only body φ labels. The 2D and 3D human pose estimation labels differ in the available labels as COCO [25], H36M [48] and SIM use different skeleton structures. Legend: ✓data is available, X data is not available, O data is partially available.

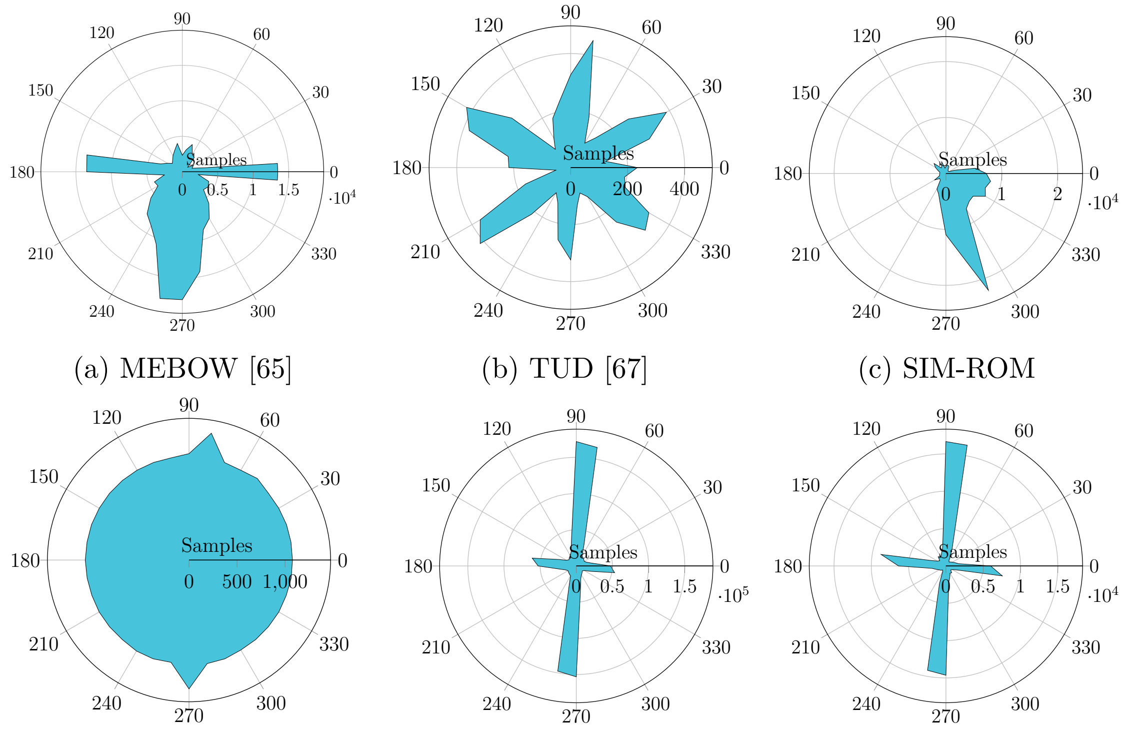
Whole-body data, including the θ and φ angle of the body and head orientations, are only available in the simulated datasets. Therefore, the validation of the orientation estimation on real data can be performed with these datasets only for the azimuthal angle φ of the body. All simulated datasets are created using motions, captured in a motion capture laboratory. For action recognition, a particular dataset, SIM-C01, was designed and recorded that is focused on pedestrian action recognition. This dataset is described in section 4.4.4.3.

4.4.4.1 SIM-ROM

In the SIM-ROM dataset, a person was recorded performing an extended range of motions that should include as many poses as possible. The idea behind this dataset is to provide data from various performable body poses for 3D human pose recognition.

4.4.4.2 SIM-CIRCLE

The SIM-Circle dataset resulted from an analysis of the other datasets. As highlighted in Figure 4.31, there is a substantial difference in the distribution of the azimuthal angle φ of the body pose.



(a) MEBOW [65] (b) TUD [67] (c) SIM-ROM

(d) SIM-CIRCLE (e) SIM-C01 (Train) (f) SIM-C01 (Val)

Figure 4.31: Distribution of body φ orientations [°] in the datasets used in this work. The plots show the distribution of the samples in a polar plot. The small peaks in SIM-CIRCLE are due to overlapping start and end frames.

Similar distribution of the azimuthal angle φ of the head poses were observed. To generate further data with a uniform distribution, the SIM-Circle dataset was created, in which 3D models walk clock- and counterclockwise in a circle at a uniform speed (see Figure 4.31d).

4.4.4.3 SIM-C01

In the previous action recognition experiments presented in Section 4.2 and 4.3, the elementary actions, ‘walk’, ‘stand’, ‘wave out’, ‘sit’, and ‘jump’ were binary classified. Since the actual actions performed by pedestrians include many more actions and combinations of these actions, a larger corpus of actions in the pedestrian context should be recorded in this experiment. In addition to a larger set of actions, multiple actions performed simultaneously, such as walking and looking around, should also be supported. The action catalog is geared for use in autonomous driving and includes, for example, hitchhiking, checking for traffic, or looking around. A list of all actions is given in Table 4.8.

Action	Frames (Train)	Frames (Val)
stand	$409,050$	$45,652$
idle	$58,206$	$10,168$
walk	$501,943$	$57,699$
jog	$235,386$	$16,207$
wave	$47,666$	$2,970$
kick a ball	$9,451$	$1,144$
throw something	$8,351$	$1,024$
looking for traffic	$131,570$	$14,819$
hitchhike	$38,288$	$4,018$
turn around	$37,380$	$3,494$
work	$34,580$	$4,263$
argue	$13,055$	961
stumble	$4,967$	837
open a car’s door	$13,342$	$1,192$
fall	$8,071$	591
stand up	$6,713$	873
fight	$15,512$	990

Table 4.8: SIM-C01 - Number of samples per action. Only frames used during the EHPI3D experiments are reported.

These actions were combined in different scenes. A scene could consist of only one action, for example, ‘hitchhike on the left sidewalk’, or of several actions in a certain sequence, for example:

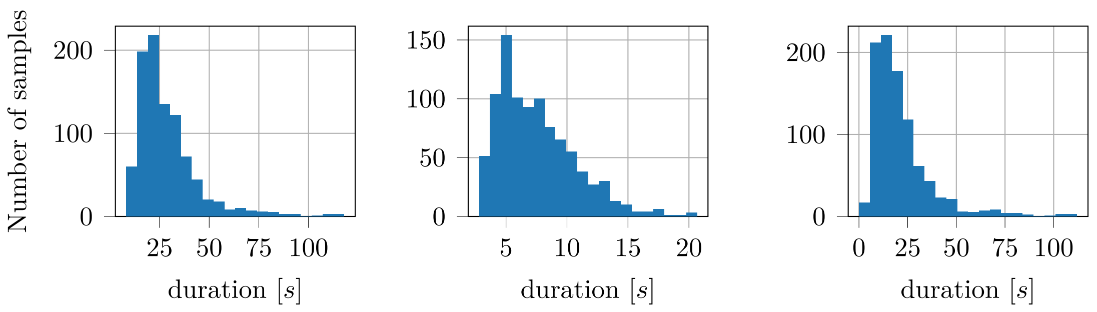
1. Walk along the left sidewalk
2. Looking for traffic
3. Cross the street
4. Turn around on the street
5. Walk back to the left sidewalk and continue there.

In addition to the instructions, various accessories, such as a smartphone, a stroller, or a walking stick, were also used during recordings (see Appendix A.3).

The SIM-C01 dataset was split in a training part (SIM-C01T) which contains the data from the first 9 of the recorded subjects and a validation part (SIM-C01V) which contains the data from the last recorded subject.

Usage of the Motion Capture Manager Since this catalog requires numerous records with complex instructions, the Motion Capture Manager (see section 3.5.1) was used for record scheduling and execution. A list of all recorded sequences as well as the number of records per sequence can be found in Appendix A.3. In this project, it was possible to evaluate how quickly complex recordings can be carried out with it. The average duration for recording a sequence ($n = 936$) was $28.84\,\mathrm{s} \pm 15.58\,\mathrm{s}$ with an average of $21.33\,\mathrm{s} \pm 15.44\,\mathrm{s}$ overhead to the actual recording, where s stands for seconds. The actual recording lasted on average $7.51\,\mathrm{s} \pm 3.12\,\mathrm{s}$. The overhead includes everything from the start of a recording to the next start of a recording, for example, the acquisition of information about the sequence, the return of used accessories, the fetching of new accessories, the movement to the start position, the preparation of the recording as well as pauses and interruptions. The recordings included between one and five instructions (mean $\mu = 2.22$, standard deviation $\sigma = 1.41$), and between zero and two accessories each in the left or right hand ($\mu = 0.96$, $\sigma = 0.74$). The ten participants were between 15 yr and 32 yr old ($\mu = 24.83$, $\sigma = 5.80$), between 166 cm and 186 cm tall ($\mu = 1.77$, $\sigma = 6.55$), and weighed between 51 kg and 115 kg ($\mu = 76.11$, $\sigma = 17.24$). Four outliers with a total length of more than 120 s were removed, as a longer pause can be assumed. Figure 4.32 shows the length of recording a sequence, the length of the captured movement, and the overhead.

The Motion Capture Manager can therefore be used to record even large recording catalogs efficiently. Due to the complete recording control by the motion capture management system, post-processing is also significantly simplified afterward since all information about all recorded sequences is directly available and can be automatically linked via the UID with the actual recording data.



(a) Record duration (b) Animation length (c) Record overhead

Figure 4.32: Motion Capture Manager: Record duration distribution

4.4.4.4 Dataset Statistics

A selection of 15 3D human models was used for all simulation datasets. The 3D models were all 3D scanned and mostly manually post-processed to ensure high quality. An overview of the 3D models used is given in Table 4.9.

Model #	Gender	Age [years]	Height [cm]	Weight [kg]	Skin [Fitzpatrick[5]]	ROM	Circle	Frames C01 (Train)	Frames C01 (Val)
1	Female	6	119	20	II	37,218	2,628	66,396	5,193
2	Female	15	168	50	V		2,639	79,923	10,037
3	Female	29	179	62	II		2,638	71,753	12,283
4	Male	29	190	85	II	36,827	2,640	62,001	8,994
5	Male	30	170	95	II		2,640	82,936	4,232
6	Female	30	173	55	VI		2,639	81,523	8,523
7	Female	30	183	65	VI	36,736	2,640	78,765	6,537
8	Male	31	187	80	III	36,948	2,568	81,490	7,601
9	Female	32	183	60	I		2,640	71,462	5,427
10	Male	40	186	75	II		2,615	52,181	9,482
11	Male	45	185	80	II		2,640	74,616	15,655
12	Male	45	185	80	II		2,640	94,833	3,267
13	Male	62	191	95	II		2,640	83,852	472
14	Male	75	182	98	I		2,639	77,537	14,722
15	Female	80	168	70	I		2,638	82,021	6,832

Table 4.9: Used 3D models and number of frames containing them in the simulation datasets.

Most models are middle-aged and have an average size. In addition to two models of seniors, one child model is also included. However, since data from particularly old, young, tall, or short people are challenging to obtain, they are underrepresented. Table 4.10 provides an overview over the datasets used. This overview shows that the H36M [48] and SIM-C01 datasets contain the most data by far. Likewise, It is clear that the H36M [48] data was recorded in a lab with limited space, which is why the bounding boxes are always relatively large. Comparing the 2D bounding box diameters of COCO [25], TUD [67], SIM-CIRCLE, and SIM-C01, the distribution is similar, as each of the datasets contains individuals at various

[5]`https://dermnetnz.org/topics/skin-phototype/` (accessed on 2022-03-06)

distances. The SIM-C01 dataset differs in that it also contains images of pedestrians taken at very far and very close distances.

Data	COCO [25]	H36M [48]	TUD [67]	SIM-ROM	SIM-CIRCLE	C01
n	$149,813$	$1,559,752$	$8,322$	$147,729$	$39,484$	$1,362,184$
$\bar{b}^{2D}$ $[px]$	229	855	193	503	222	214
σ_b^{2D} $[px]$	148	83	44	502	133	196
$\min b^{2D}$ $[px]$	28	585	45	31	43	9
$\max b^{2D}$ $[px]$	$1,002$	$1,339$	462	$2,202$	830	$2,201$
$\bar{b}^{3D}$ $[mm]$		$1,712$		$1,816$	$1,885$	$1,855$
σ_b^{3D} $[mm]$		197		401	194	230
$\min b^{3D}$ $[mm]$		859		169	$1,202$	78
$\max b^{3D}$ $[mm]$		$2,769$		$2,840$	$2,194$	$2,696$
$\bar{d}$ $[px]$		$5,170$		$7,205$	$10,609$	$16,134$
σ_d $[px]$		750		$5,578$	$5,779$	$5,498$
$\min d$ $[px]$		$2,530$		22	$2,321$	$1,640$
$\max d$ $[px]$		$7,690$		$21,289$	$25,754$	$76,657$

Table 4.10: Statistics of datasets used in the PedRec experiments. b represents the 2D or 3D bounding box size, and d the distance to the camera. The number of samples is notated as n, the mean of bounding boxes and the distances to the camera as $\bar{b}/\bar{d}$, and the standard deviation values are notated with σ.

4.4.5 Training Procedure

The network was trained step by step in the following order:

1. 2D human pose estimation
2. 3D human pose estimation
3. Joint visibility
4. Head and body orientation

The training was performed on real data (COCO [25]+H36M [48]) and then on simulation data in the same order.

Loss Functions The L_1 loss function was used for the 2D and 3D human joint coordinate regression losses L_{p2d} and L_{p3d}. The joint visibility loss L_{vis} is the standard binary cross-entropy loss. In the orientation regression task, circular data was represented in a one-dimensional map. Thus, the standard L_1 or L_2 loss could not be used. For example, a prediction of 359° with a ground truth of 0° results in an error of 359°. This applies only for the azimuthal angle φ, the polar angle θ is defined between 0° and 180°, and thus the standard distance metrics can be used. As such, the following loss functions were applied:

$$L_\varphi = \frac{\sum_{i=1}^{N} \min\left(1 - |\hat{m_\varphi} - m_\varphi|, |\hat{m_\varphi} - m_\varphi|\right)}{N} \tag{4.19}$$

$$L_\theta = \frac{\sum_{i=1}^{N} |\hat{m_\theta} - m_\theta|}{N} \tag{4.20}$$

$$L_o = \frac{L_\varphi + L_\theta}{2} \tag{4.21}$$

where N is the number of samples and m_φ and m_θ are the softargmax outputs for the azimuthal angle φ and the polar angle θ normalized between zero and one. As shown in equation 4.19 and 4.20 the L_1 loss is incorporated. For training data which only provides labels for the azimuthal angle φ only equation 4.19 is used.

In the experiments, N may be a subset of the entire training set, as various datasets with different supported labels (see table 4.7) were combined. As such, each loss function contains a sample selection step before the actual calculation of the loss.

Uncertainty loss, described by Kendall *et al.* [138], was applied to balance the different loss outputs. The final loss function is:

$$L = \frac{1}{2\sigma_1^2} L_{p2d} + \frac{1}{2\sigma_2^2} L_{p3d} + \frac{1}{2\sigma_3^2} L_o + \frac{1}{\sigma_4^2} L_{vis} + \log\left(1 + \prod_{i=1}^{4} \sigma_i\right) \tag{4.22}$$

where σ_{1-4} are learnable parameters. $\log(1+\sigma)$ was used instead of $\log(\sigma)$ to ensure a positive loss value. Using this approach resulted in slightly better performance compared to equal-weighted losses. The training procedure remained stable using both approaches.

Optimizer The AdamW optimizer [139] was used for optimization, which is a slightly modified variant of the Adam optimizer [140]. The learning rate range test [141] was applied to get an initial learning rate of $4e^{-3}$. A standard weight decay of $1e^{-2}$ was used. In addition, the 1cycle policy [142] was used, with which the learning rate is updated during the training process from a minimum learning rate of $\frac{4e^{-3}}{25}$ to the maximum of $4e^{-3}$ and afterward back to a minimum using cosine annealing. Smith showed that this approach results in faster convergence and usually better results [141]. The network was trained with a training cycle of 15 epochs, from which 10 epochs were trained with frozen weights in the feature extractor. For the last five epochs with the feature extractor unfrozen, the learning rate for the feature extraction was reduced to $2e^{-4}$ and for the other layers to

$4e^{-4}$. The training cycle was repeated five times, which improved the performance slightly.

Datasets The training datasets of COCO [25], H36M [48], SIM-ROM, and SIM-Circle were used to train the PedRecNet. The validation is done on the validation parts of COCO [25], H36M [48], and the SIM-C01 dataset. The H36M [48] training dataset was subsampled by ten, all samples from the other datasets were used. For the orientation estimation part, only labels from SIM-ROM and SIM-Circle were used during the initial training. COCO [25] labels were used in an additional training step during orientation experiments (see section 4.4.6.3). The dataset names are abbreviated in the results as follows: C stands for COCO [25], M for COCO [25] (MEBOW [65]), H for H36M [48] and S for SIM-Circle and SIM-ROM combined. COCO [25] is always the base dataset of PedRecNet and is therefore used as one of the training dataset in every experiment.

Augmentations The data were augmented by scaling the inputs by up to ±25% and rotating them by up to ±30° in 50% of the cases, but only when no orientation labels were used. The data were flipped in 50% of the cases.

4.4.6 Results

In the following evaluation, the results of 2D and 3D human pose recognition and orientation estimation are considered. The focus of this evaluation is to determine the influence of data from the different datasets on the results. This is in particular the case for the use of the simulated data. Comparisons with SOTA methods are included to provide a general performance ranking to determine whether PedRecNet is generally suitable as a baseline against specialized networks.

4.4.6.1 2D Human Pose Estimation

We first look at performance based on results on the COCO [25] val2017 benchmarks. For comparability, all approaches, except the bottom-up approach OpenPose, were compared using ground truth bounding boxes. The results are shown in Table 4.11.

As expected, the performance between the PedRecNet, when trained only on the COCO [25] dataset, is comparable to the approach of Xiao *et al.* [30] on which the network is based. The differences in performance can be explained by an extended test-time augmentation or post-processing in the approach of Xiao *et al.* [30]. More current approaches, such as the HRNet [143], deliver significantly better performance. Thus, it might be worth changing the backend from a ResNet50 to an HRNet [143] in the future. Additionally, results for the PedRecNet trained on COCO [25] only using test-flip augmentation are provided. The test-time flipping

Datasets	AP	AP^{50}	AP^{75}	$AP(M)$	$AP(L)$	AR	AR^{50}	AR^{75}	$AR(M)$	$AR(L)$
Cao *et al.* '17 [27]	65.3	85.2	71.3	62.2	70.7	-	-	-	-	-
Xiao *et al.* '18 [30] (ag*)	72.4	91.5	80.4	69.7	76.6	75.6	93.0	82.3	72.4	80.4
Sun *et al.* '19 [143] (ag*)	76.5	93.5	83.7	74.0	80.8	79.3	94.5	85.8	76.2	84.1
ours C (2D only) (ag)	71.5	91.4	78.7	68.4	76.1	75.0	92.3	81.1	71.7	80.0
ours C (2D only) (g)	70.4	91.3	77.6	67.6	74.9	74.2	92.0	80.5	71.0	79.0
ours C+H+M (g)	68.8	90.3	76.6	65.9	72.9	72.6	91.2	79.4	69.5	77.3
ours C+S (g)	67.9	90.3	75.0	64.9	72.8	72.3	91.5	79.0	68.9	77.5
ours C+H+S (g)	67.1	90.3	74.9	64.6	71.4	71.5	91.1	78.4	68.3	76.3
ours C+H+S+M (g)	67.6	90.2	75.1	65.0	71.9	71.8	91.4	78.6	68.7	76.7

Table 4.11: Comparison of used training datasets on COCO val2017 results. Ground truth bounding boxes were used as detection input. Dataset Legend: C=COCO [25], H=H36M [48], M=MEBOW [65], S=SIM, (a) used test-time augmentation (in ours only flip test is applied), (g) used ground truth bounding boxes, *= the officially provided code was used to evaluate on GT bounding boxes. 192×256 input bounding boxes and the ResNet50, respectively the W32 architecture were used.

improves accuracy on validation data, which is more or less pronounced depending on the architecture. However, since this leads to double the size of the input and the usefulness of this method under real conditions is questionable, further 2D results are presented without any test-time augmentations. In the present work, it is more relevant to look at the relative differences between the used datasets on the PedRecNet. Here, it can be observed that under the additional use of the H36M [48] and MEBOW dataset [65], the AP decreases by 0.6%. If the simulation dataset is also used, the performance decreases by another 1.2%. Looking at AP^{50} versus AP^{75}, it is clear that the performance differences have little effect at an OKS of >= 0.5, but become more pronounced at >= 0.75. One explanation for the performance loss would be slight differences in the label positions of the training data used (see discussion in section 2.3.3.1, which also applies for the OKS metric).

Table 4.12 shows the results using the PCK metric on the COCO [25], H36M [48], and SIM-C01 datasets.

The PCK results on the COCO [25] val2017 dataset and the conclusions to be drawn are comparable to the OKS results on the COCO [25] val2017. Going further, it is noticeable that the network trained purely with COCO [25] already delivers relatively good results on the SIM-C01V dataset, while performing rather poorly on H36M [48], which only contains individuals at a short distance from the camera. One factor influencing the performance on H36M may be the difference in the ground truth label positions. However, it is also important to note that in the H36M [48] dataset, there is no ground truth to the joints on the face, so the comparison is only made on the 12 remaining joints. The H36M [48] results on the full PedRecNet dataset improve slightly with the use of the simulation data. The

Net	Train	C@0.05	C@0.2	H@0.05	H@0.2	S@0.05	S@0.2	H-MPJPE	S-MPJPE
2D	*C*	56.47	92.65	31.25	88.46	48.70	96.06	–	–
2D	*C+H*	55.26	92.47	58.44	93.07	48.88	96.14	–	–
2D	*C+S*	54.53	92.19	31.84	88.55	64.56	96.55	–	–
Full	*C+H*	54.99	92.19	60.74	93.24	49.76	96.37	63.96	158.0
Full	*C+H+S*	53.18	91.87	61.66	93.34	65.64	96.67	64.54	91.8

Table 4.12: 2D Human Pose Estimation Results: COCO [25] Skeleton (17 joints, see section 3.7.4.1). Note that the H36M [48] dataset does not contain joints for the face, thus the evaluation on H36M [48] is done only on the remaining 12 joints.

fully trained network achieves good performance on all validation datasets, with *PCK*@0.05 exceeding 50% and *PCK*@0.2 exceeding 90%. Overall, 2D human pose estimation is noticeably better on simulation data, although long-range data is included. However, this can also be explained through the usage of a relatively small corpus of 3D human models, which occur accordingly in training as well as in the test dataset, albeit with different motions.

It is important to note that the skeletal structure differs in the datasets, so the evaluation of 2D human pose estimation must also be considered on the official H36M [48] skeletal structure and the PedRec skeletal structure. Table 4.13 shows the results with the H36M [48] skeletal structure.

Net	Train	H@0.05	H@0.2	S@0.05	S@0.2
Full	*C+H*	89.2	98.1	76.3	98.8
Full	*C+H+S*	89.4	98.1	91.2	98.9

Table 4.13: 2D Human Pose Estimation Results (PCK scores): H36M [48] skeleton with 17 joints (see section 3.7.4.1).

The results are consistently much better. This is mainly because the bounding boxes in the H36M [48] dataset are larger, thus providing better visual cues. Differences between the datasets used here only affect the details; for example, PCK@0.05 is about 15% better on the SIM-C01 dataset, but PCK@0.2 is virtually identical, again more indicative of differences in ground truth label positions. Table 4.14 shows the results on the extended H36M [48] skeleton which includes hand and foot ends in addition to the other 17 joints. These joints are included in the dataset but are not part of the official skeletal structure (see [48, p. 1335]).

The overall PCK decreases when including the foot and hand ends, but is still at 86.3% for *PCK*@0.05. Accordingly, we can conclude that recognizing the hand and foot positions works, but worse than the other joints. As in the previous experiments, the result on the H36M [48] dataset is slightly better using the simulation

Net	Train	H@0.05	H@0.2	S@0.05	S@0.2
Full	$C{+}H$	86.1	97.5	70.5	98.2
Full	$C{+}H{+}S$	86.3	97.6	87.8	97.9

Table 4.14: 2D Human Pose Estimation Results (PCK scores): H36M extended skeleton with 21 joints (H36M [48] skeleton + HAND and FOOT ends.).

data. Interestingly, the PCK@0.2 result on the simulation dataset worsens slightly when training with simulation data compared to training with H36M [48] data only. However, due to the slight difference and the generally high level, this can also be assumed to be a simple training effect.

Table 4.15 shows the results on the full PedRec skeleton.

Net	Train	S@0.05	S@0.2
Full	$C{+}H$	38.0	94.5
Full	$C{+}H{+}S$	64.1	95.8

Table 4.15: 2D Human Pose Estimation Results (PCK scores): PedRec skeleton with 26 joints (see section 3.7.4.1).

The PedRec skeleton is a combination of the H36M [48] extended and the COCO [25] skeletons. As expected, the performance is somewhat worse than the evaluation with the H36M [48] extended skeleton, since the 2D joints in the face are generally more challenging to estimate.

Figure 4.33 show an example on real 'in the wild data'. Generally, good detection performance is achieved, but some false detections are also visible. The detection of the distant humans already fails during the detection with the used YoloV4 network. One example of false detections in the 2D human pose estimation is the left foot end of the third man from the left, partially obscured by the red jacket. The same applies to the woman in the middle, whose right foot is occluded by the left leg. Another example is the wrist as well as the end of the hand of the man on the far right, who is scratching his head. Occlusions thus still seem to cause problems for the network in some cases.

4.4.6.2 3D Human Pose Estimation

For 3D human pose estimation, first the performance compared to other methods on the H36M [48] validation dataset is reported. The results of 3D human pose estimation are shown in Table 4.16. It summarizes further approaches, which differ in the methods, input data, and validation data used. Some approaches use test-time augmentations like flip-testing or temporal information to improve the results.

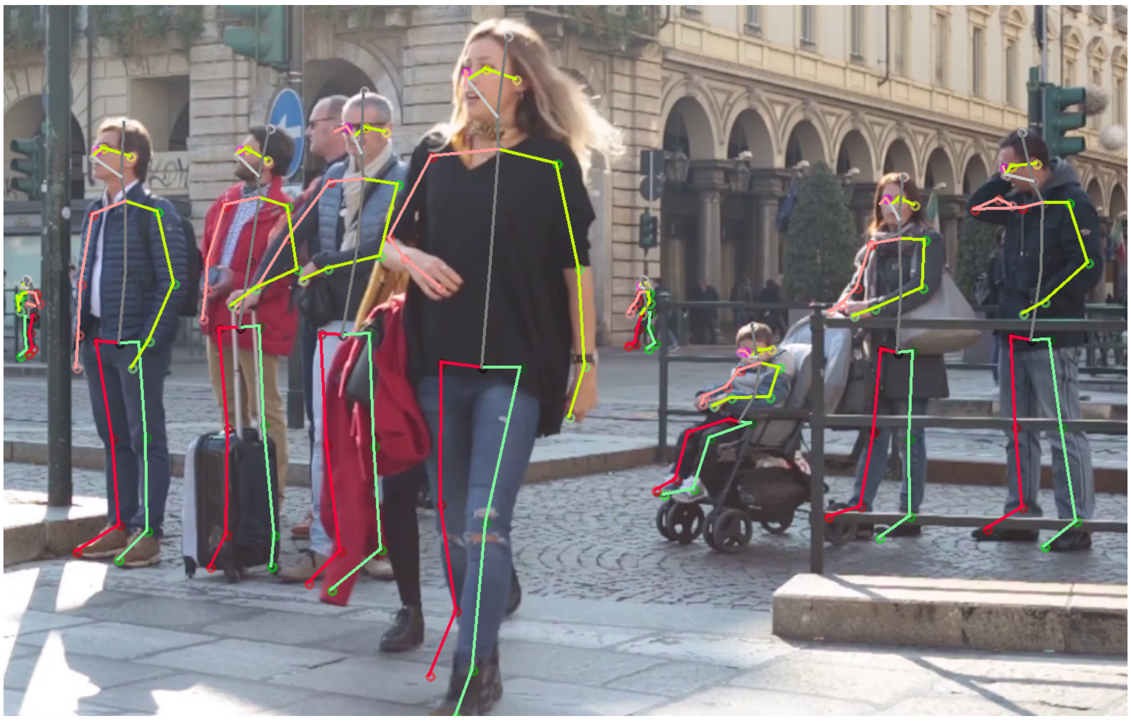


Figure 4.33: 2D human pose estimation: Real-world example. Subjectively good results, yet occasionally a few problems with occlusions and foot / hand end positions.

Method	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	SitD.	Smoke	Wait	WalkD.	Walk	WalkT.	Avg
Chen *et al.* '17 [38](g)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	82.4
Martinez *et al.* '17 [46](g)	51.8	56.2	58.1	59.0	69.5	78.4	55.2	58.1	74.0	94.6	62.3	59.1	65.1	49.5	52.4	62.9
Luvizon *et al.* '17 [137] (gs)	49.2	51.6	47.6	50.5	51.8	48.9	48.5	51.7	61.5	70.9	53.7	60.3	44.4	48.9	57.9	53.2
Yang *et al.* '18 [144]	51.5	58.9	50.4	57.0	62.1	65.4	49.8	52.7	69.2	85.2	57.4	58.4	43.6	60.1	47.7	58.6
Pavllo *et al.* '19 [39](agt)	45.1	47.4	42.0	46.0	49.1	56.7	44.5	44.4	57.2	66.1	47.5	44.8	49.2	32.6	34.0	47.1
Pavllo 2018 *et al.* '19 [39](ag)	47.1	50.6	49.0	51.8	53.6	61.4	49.4	47.4	59.3	67.4	52.4	49.5	55.3	39.5	42.7	51.8
Luvizon *et al.* '20 [37](gs)	43.2	48.6	44.1	45.9	48.2	43.5	44.2	45.5	57.1	64.2	50.6	53.8	40.0	44.0	51.1	48.6
Shan *et al.* '21 [145](at)	40.8	44.5	41.4	42.7	46.3	55.6	41.8	41.9	53.7	60.8	45.0	41.5	44.8	30.8	31.9	44.3
Gong *et al.* '21 [146](ag)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	50.2
ours C+H+M (ag)	49.2	51.9	49.6	50.9	55.4	60.4	45.4	48.8	64.3	75.2	53.0	47.3	54.0	39.2	45.5	52.7
ours C+H+S+M (ag)	51.2	51.9	49.7	52.1	56.0	60.3	47.1	48.8	62.8	75.6	53.1	48.2	54.2	40.5	47.3	53.3
ours C+H+M (g)	50.4	53.8	50.8	52.9	57.5	62.9	47.2	50.5	65.8	78.6	54.8	49.2	56.0	40.7	46.3	54.5
ours C+H+S+M (g)	52.4	53.9	50.8	54.0	57.8	62.6	48.6	50.3	64.6	79.0	54.7	49.6	56.3	41.6	48.1	54.9

Table 4.16: Results on the H36M [48] dataset reported as MPJPE. Legend of properties (Prop.), influencing the results: (a) used test-time augmentation (in ours only flip test is applied), (g) used ground truth bounding boxes, (s) sampled every 64th frame of validation set, (t) used temporal information

For better comparability, results using flip-testing are shown in addition to the results without any form of test-time augmentation. The performance of our method with an average MPJPE of 52.7 mm is comparable to current SOTA approaches such as Gong *et al.* [146] with 50.2 mm and Luvizon *et al.* [37] with 48.6 mm. The use of temporal information leads to noticeable better results in this benchmark. This becomes clear when comparing the two results of Pavllo *et al.* [39] who have been able to achieve an improvement of 4.7 mm by using temporal information. The performance from PedRecNet is very similar, but decreases slightly when simulation data is used during training. However, this may also be because the simulation data is partly very different from the H36M [48] data in terms of distances, body size of the persons and label positions.

An even better overview of the actual performance is provided by looking at the Δ between predicted and ground truth joint position per joint separately. Figure 4.34 shows these for the H36M [48] dataset and the fully trained PedRecNet. Joints near the body center are detected much better than the extremities. This was to be expected since the extremities have an increased degree of freedom as well as often a greater distance from the pelvis. In addition to the MPJPE, the joint distance is shown in x, y, and z direction. The most significant contribution to the error comes from depth estimation.

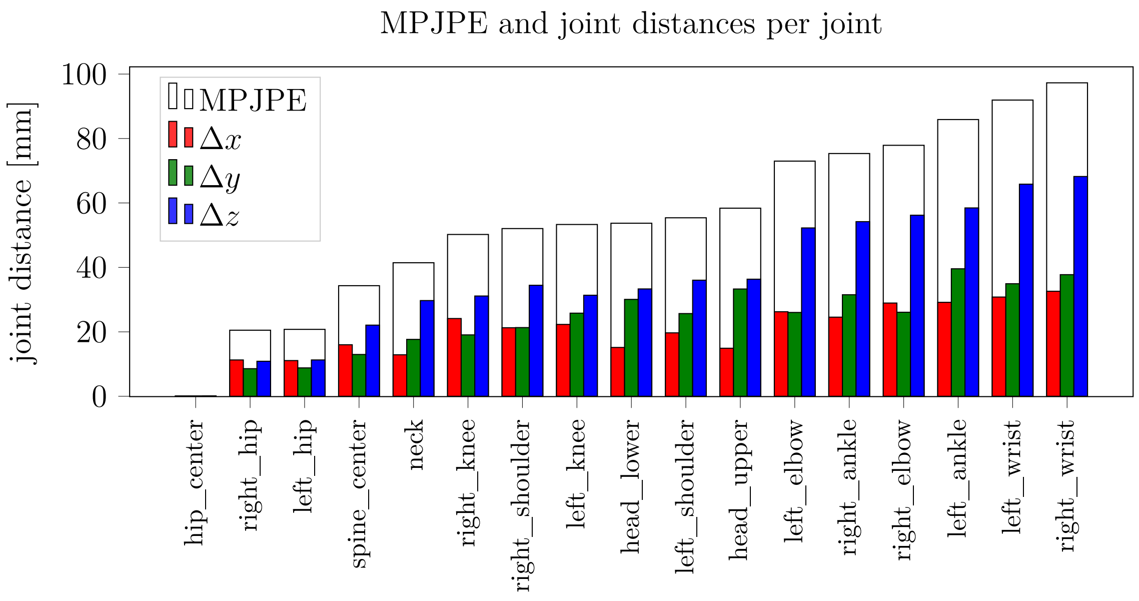


Figure 4.34: 3D joint distances between prediction and ground truth on the H36M [48] validation dataset. Results reported using PedRecNet trained on C+H+S+M. The colored bars show the Δx, Δy and Δz of the predicted and the ground truth joint position. The containing white bar shows the MPJPE for the given joint.

The results for the same evaluation on the SIM-C01 validation dataset with all 26 joints are shown in Figure 4.35. We see similar results with the most significant errors for the extremities. The most negligible errors occur for joints near the centerline of the body. This also includes the joints on the face. It is interesting to note that on the SIM-C01 validation dataset, the relative contribution of the depth estimate to the total error for extremities is smaller than when evaluated on the H36M [48] dataset. This may be because the SIM-C01 validation dataset includes more different body sizes and more variations in distance, making pose estimation more difficult in the x and y directions.

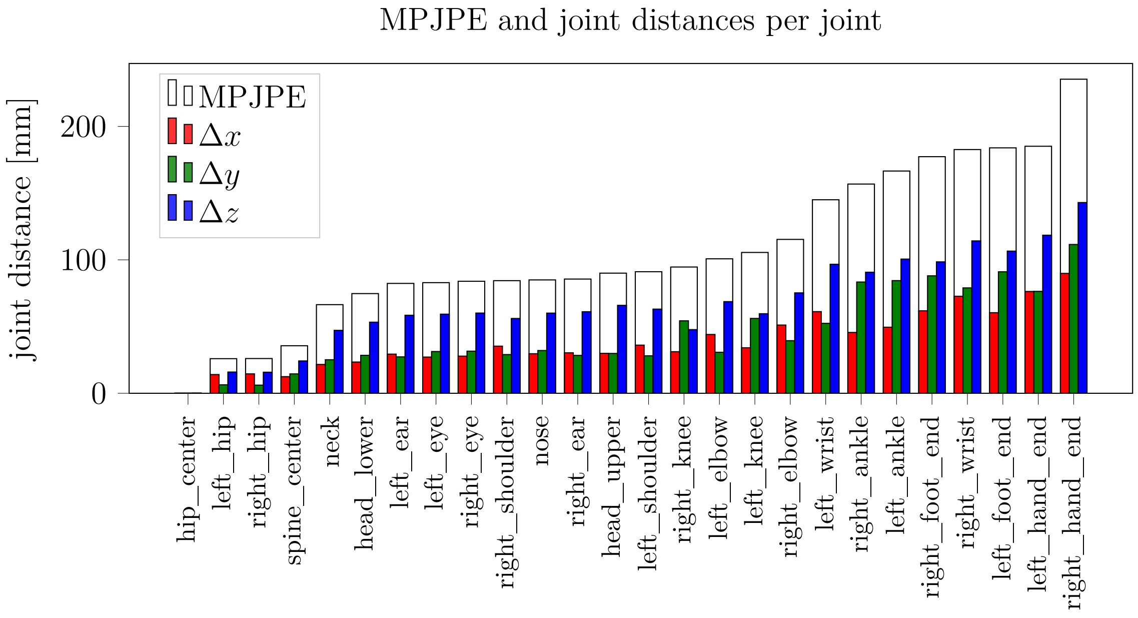


Figure 4.35: 3D joint distances on the SIM-C01 validation dataset. Results reported using PedRecNet trained on $C{+}H{+}S{+}M$.

Figure 4.36 shows some examples on 'in the wild' real data. The examples are from various sources and include different cameras, focal lengths, exposures, and perspectives. Examples 4.36d-4.36f show that even in challenging situations a good 3D human pose can be predicted.

In contrast, some error cases are demonstrated in Figure 4.37. Figure 4.37a shows an extreme corner case where the pose estimation fails completely. Note that the corner case dataset from section 4.1 was not used during training. In the second example, the body pose is roughly correct, but still shown as too upright. The examples 4.37c and 4.37d show people with their arms stretched out, and the left arm is estimated incorrectly. The same applies to the example 4.37e. The pose is correct in most parts, but the outstretched left arm is not correctly

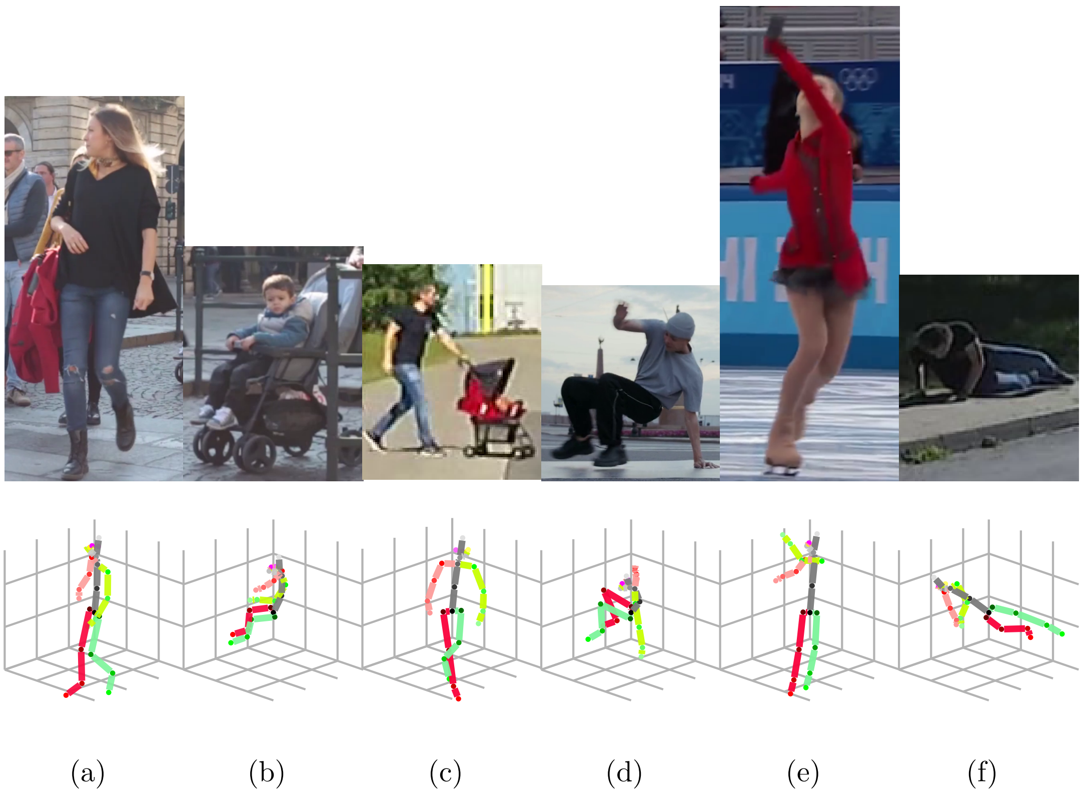


Figure 4.36: 3D Human pose estimation ‘in the wild’: PedRecNet examples. Top: Cropped image of the person inputted in the network. Bottom: Predicted 3D human pose.

recognized. From the pose only, it is not clear that the person is just operating a traffic light switch. Example 4.37f shows a false recognition due to self occlusion. The body occludes the left arm, but the 3D human pose recognition estimates it to be hidden behind the right arm and displays it stretched out accordingly. These false detections by occlusion could possibly be improved by further training data or the use of temporal context.

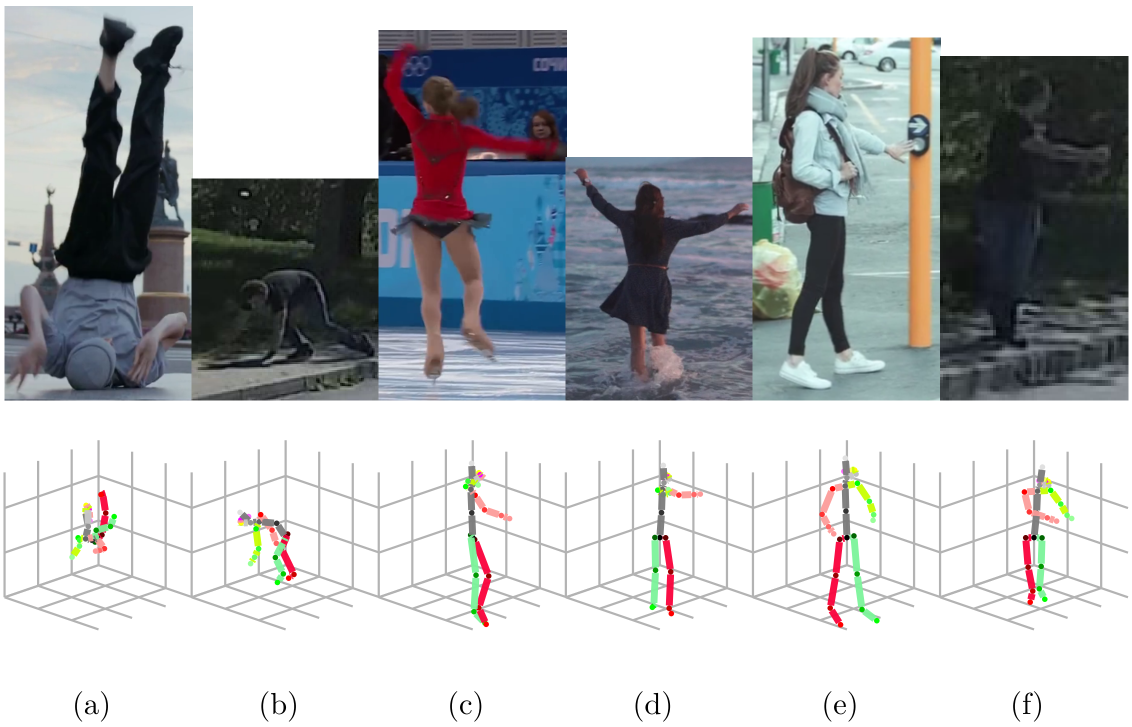


Figure 4.37: 3D Human pose estimation in the wild: PedRecNet false predictions. Top: Cropped image of the person inputted in the network. Bottom: Predicted 3D human pose.

4.4.6.3 Body Orientation Estimation

As described in the related work section, there are only few datasets in the context of body and head orientation estimation for full-body inputs. For a state-of-the-art comparison, the relatively new dataset MEBOW [65] is suitable. This dataset provides body orientation labels for the COCO [25] dataset. However, it only contains the azimuthal angle φ, labels for the head pose are not included. The TUD [67] dataset was also used in the analysis, although it only contains 309 samples in the validation dataset. Accordingly, the significance of the results on the TUD validation data is relatively low. Table 4.17 gives an overview over the results on these datasets.

Network	Trainset	Testset	*Acc.*(22.5°)	*Acc.*(45°)	MAE(°)
Wu *et al.* [65] (2020)	MEBOW	MEBOW	**93.9**	**98.2**	8.4
ours	MEBOW	MEBOW	**92.3**	**97.0**	9.7
ours	SIM+MEBOW	MEBOW	91.7	**97.0**	10.0
ours	SIM	MEBOW	80.2	94.7	16.1
Hara *et al.* [147] (2017)	TUD	TUD	70.6	86.1	26.6
Yu *et al.* [148] (2019)	TUD	TUD	75.7	96.8	15.3
Wu *et al.* [65] (2020)	MEBOW	TUD	77.3	99.0	14.3
ours	MEBOW	TUD	**79.6**	**99.0**	10.8
ours	SIM+MEBOW	TUD	77.3	98.7	14.3
ours	SIM	TUD	**75.4**	**98.1**	16.0
ours	MEBOW	SIM-C01	76.2	97.0	16.6
ours	SIM+MEBOW	SIM-C01	**79.7**	**97.9**	15.3
ours	SIM	SIM-C01	78.7	96.5	16.0

Table 4.17: Human body orientation (φ) test results on the MEBOW [65], TUD [67] and SIM-C01V datasets. The column trainset specifies the training dataset(s) used to train the specific networks. Testset specifies on which test dataset the results are reported on. In addition to the accuracy in 22.5° and 45° intervals, the mean average error (MAE) is reported.

It is to notice that the PedRecNet already achieved an *Acc.*(22.5°) of 75.4% on the TUD [67] dataset and 80.2% on the MEBOW [65] dataset. For *Acc.*(45°), which is often sufficient for real-world applications, it even achieves 98.1% on the TUD [67] dataset and 94.7% on the MEBOW [65] dataset. These are surprising results for not using any training data from the corresponding training datasets. Especially when compared to the earlier approaches of Hara *et al.* [147] and Yu *et al.* [148], the PedRecNet gives better results without ever having seen any data from the TUD [67] dataset. Accordingly, the PedRecNet provides a solid baseline, trained with simulated data only. However, it should be noted, that 3D human pose data is also used for orientation estimation and the training for this has included real data from the H36M [48] dataset. When the MEBOW [65] training data are used in addition to the simulation data, the *Acc.*(22.5°) and *Acc.*(45°) improve by 11.5% and 2.3%, respectively, and are 2.2% and 1.2% worse than the results reported by Wu *et al.*. In total, 159 body orientations were predicted with an error above 45°. These misclassifications were analyzed further, and erroneous ground truth labels were detected for 37 images, some of which are shown in Figure 4.38.

Further, in 44 images, occlusions by different persons and thus confusions of the detected person are present, examples of which are shown in Figure 4.39a-4.39c. 22 images contain images of persons that, even for humans, are very difficult to see. Some examples of these problems are shown in Figure 4.39d-4.39f.

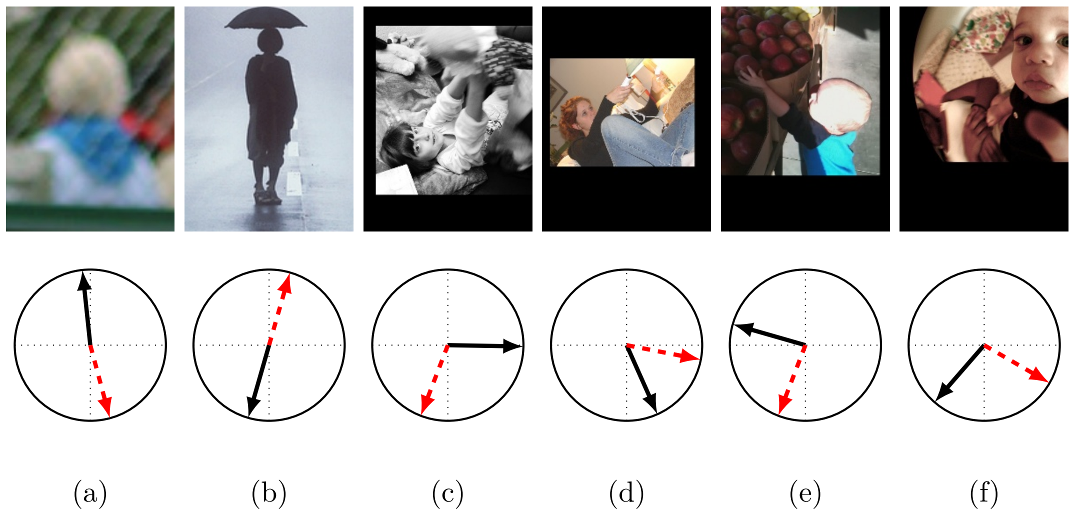


Figure 4.38: MEBOW [65] validation dataset: Examples of wrong orientation labels. The red dotted arrow shows the annotated ground truth, the black arrow shows the prediction of PedRecNet. The annotation of people in a not upright position, for example 4.38c, is debatable.

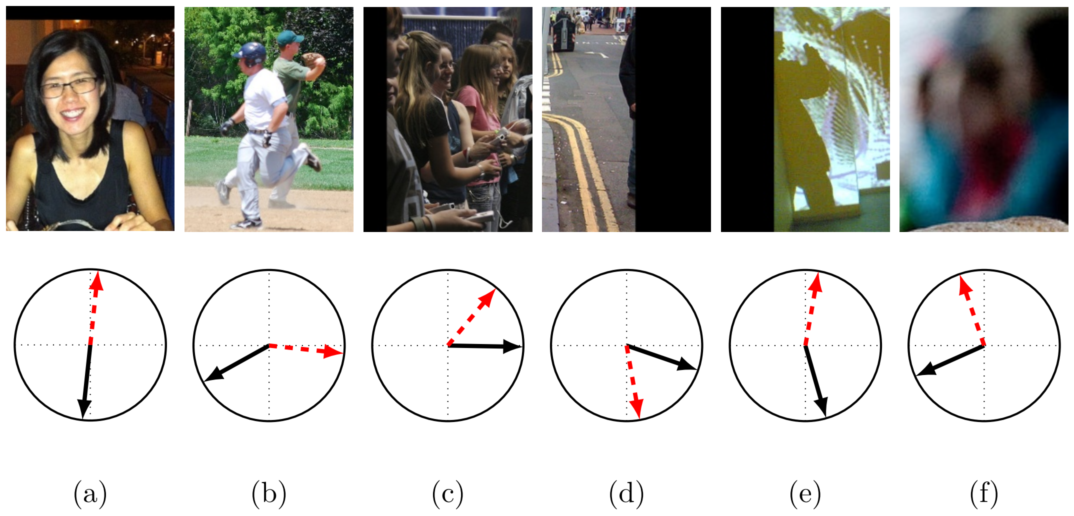


Figure 4.39: MEBOW [65] validation dataset: Examples of pictures of crowded or occluded scenes, where it is not directly clear, which person should be annotated (4.39a-4.39c) and examples of pictures where the person is hard or not to see (4.39d-4.39f). The red dotted arrow shows the annotated ground truth, the black arrow shows the prediction of PedRecNet.

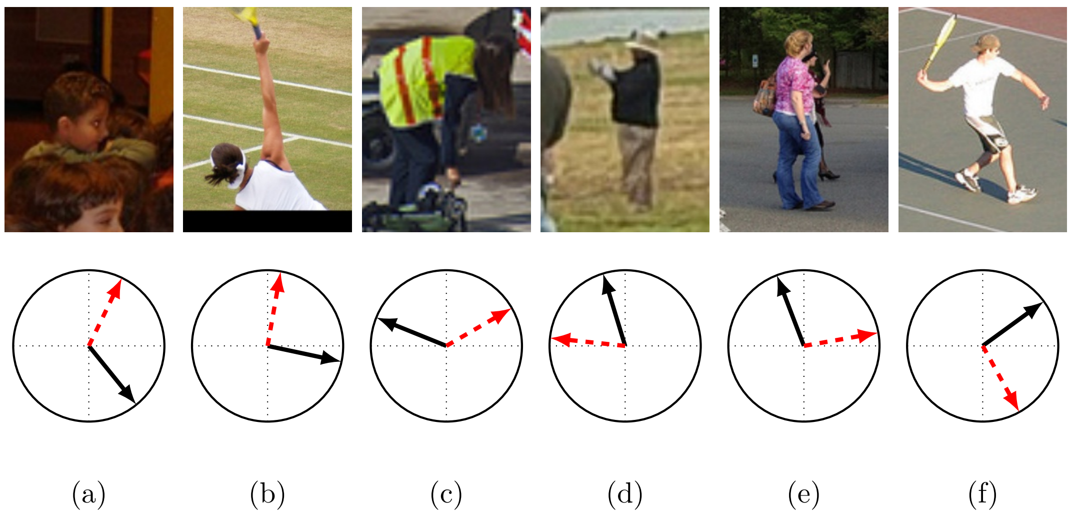


Figure 4.40: MEBOW [65] validation dataset: Examples of wrong orientation predictions. The red dotted arrow shows the annotated ground truth, the black arrow shows the prediction of PedRecNet.

The remaining 65 images are clear false detections of the PedRecNet, examples of which are shown in Figure 4.40.

4.4.6.4 Head Orientation Estimation

The SIM-C01V dataset was used for the validation of the head orientation (φ) estimation. Only the φ estimate is considered at this point because θ is underrepresented in the SIM-C01 dataset; the head orientations are relatively horizontal in the pedestrian actions in almost all cases. Accordingly, for the estimation of θ, further targeted experiments and new data recordings are needed in the future. The results for the estimation of the head φ orientation are shown in Table 4.18.

Network	Trainset	Testset	*Acc.*(22.5°)	*Acc.*(45°)	*MAE*
PedRec	SIM+MEBOW	SIM-C01V	77.1	95.1	16.65
PedRec	SIM	SIM-C01V	76.3	94.8	17.43

Table 4.18: PedRecNet: Head orientation test results for φ.

The results are slightly inferior to the body orientation estimation by 2.6% and 2.8% for *Acc.*(22.5°) and *Acc.*(45°), respectively. In general, however, performance on the body and head orientation estimates is relatively similar. The somewhat inferior performance can be explained by the head region's smaller image area than the body region. The author is not currently aware of a larger and publicly available dataset that includes head orientation images in addition to full-body images.

Therefore, most approaches to head orientation estimation work with datasets that only contain cropped faces. In productive applications, face recognition can then be performed first, followed by a crop of the face, and orientation estimation can be performed based on this cropped face bounding box. In addition, most datasets only contain faces, which means that a side view or the back of the head cannot usually be used for orientation estimation. In our approach, the entire body is always considered, which enables head orientation estimation even for a side and back view of a person. However, subjective observation of 'in-the-wild' examples hint, that similar performance is expected on real data for head pose recognition as for body pose recognition when trained on simulation data only. Figure 4.41 shows some 'in-the-wild' examples. Figure 4.41a shows a typical example, where one can nicely depict the different estimates of head versus body orientation. Example 4.41b shows a boy in a stroller, which shows that the orientation estimation gives good results even in non-upright positions. Figure 4.41c shows a person who was photographed from behind. Especially the correct head pose estimation is interesting, although the person wears a hood and only a small part of the nose is visible. Another interesting example is demonstrated in Figure 4.41d, where the orientation estimation is based on input data of a person shot from behind and only visible in a low-resolution image section of about $38 \times 78px$.

4.4.7 Discussion

The presented PedRecNet is a simple yet efficient architecture that performs multiple tasks simultaneously and can run on consumer hardware at over 15FPS even with multiple people. The network achieves performance that is comparable to current SOTA methods for 2D and 3D human pose estimation and orientation estimation. The presented model combines all these tasks in a simple and extensible architecture which is straightforward to train. Thus, the introduced model is also well suited as a baseline for further research. It was further shown that the orientation estimation part can be trained purely with simulation data and achieve high accuracy on real data without requiring real sensor data for training. Further, the 3D human pose estimation of PedRecNet can be used for the following EHPI3D action recognition experiments.

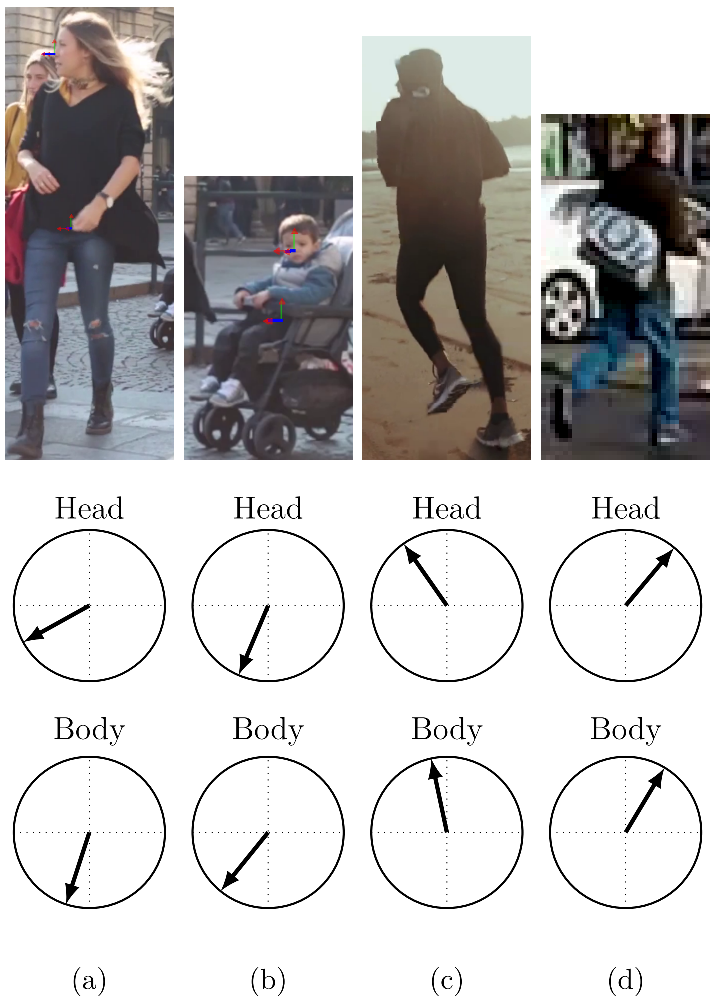


Figure 4.41: Head and body orientation estimation 'in the wild': Examples. Top: Cropped image of the person processed by the network. Bottom two rows: Predicted head and body orientation φ.

4.5 EHPI3D action recognition framework based on 3D human pose

The detection of pedestrians, as well as their behavior, remains a challenge in the field of autonomous driving. As shown in section 4.2 it is possible to detect pedestrian actions using 2D human pose recognition data, but the applied method is susceptible to the vehicle's own motion due to the usage of image pixel coordinates of the front-facing camera in the car. In this work, pelvis-aligned 3D human pose data are used as input to the action recognition, which are correspondingly independent of camera movements. The following chapter will describe how the EHPI approach is extended to 3D and show in experiments how action recognition can be performed with this data. A novel dataset, called SIM-C01, is presented for this purpose. This dataset contains significantly more actions than the experiments described in sections 4.2 and 4.3. It includes various, selected actions relevant to traffic situations and not available in other public datasets yet. Multi-label action classification is used in this dataset and the following experiments.

The entire system and novel simulation data has been made public under the MIT license[6].

4.5.1 Generation of EHPI3Ds

The basic generation of an EHPI3D is similar to the generation of an EHPI2D presented in section 4.2. The three RGB color channels encode the x, y, and z joint coordinates retrieved from the 3D human pose estimation part of the PedRecNet. These coordinates are placed in an $M \times N \times C$ matrix, where M represents the individual human joints, N represents the temporal dimension, and C represents the three color channels. This structure allows the EHPI3D to be visualized as an image, just like the EHPI2D. A matrix of size $32 \times 32 \times 3$ covers a period of 1.06 seconds for a video input at 30 FPS. Likewise, up to 32 joints are supported. The complete PedRec skeleton uses only 27 joints of this. These are raised to 32 with zero padding. The input size was chosen because all common standard classification networks can be used with it. Alternatively, it would also be possible to set the input size to $27 \times 32 \times 3$ and pad the data during runtime. N is also padded with zeros if there is not enough data from previous frames. No further adjustment is required since the 3D joint coordinates are already normalized in a $3 \times 3 \times 3$ meter volume (see Figure 4.21). This is also a major advantage over the EHPI2D approach. By using image coordinates, the EHPI2D is usually influenced by the camera's ego-motion. To overcome this issue, one would need to use methods like optical flow to estimate the ego-motion and remove it from the EHPI2D. The 3D joint positions, on the other hand, are in the local coordinate system of the

[6]`https://github.com/noboevbo/PedRec` (accessed on 2022-03-06)

observed human and are therefore independent of the camera's movements. Thus, the EHPI3D approach can be used on moving platforms. However, the size of the humans is not normalized, so the EHPI3D images of the same motions differ for humans of different size. To compensate for this effect, the 3D joint positions are normalized using a unit skeleton (see section 4.4.1). The difference in performance after this normalization is shown in section 4.5.4. Figure 4.42 shows the structure of an EHPI3D, which mainly groups nearby human joints.

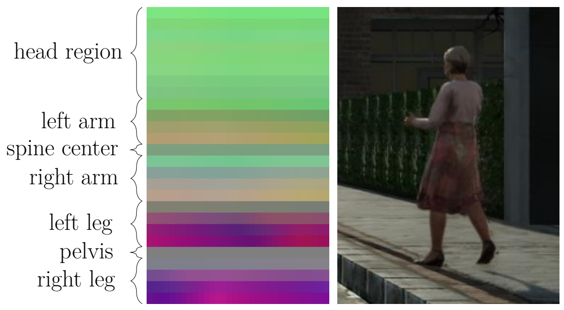


Figure 4.42: Example of the EHPI3D structure (left) with the last frame of the inputted data (right) of the action 'walk'

.

4.5.2 Unit Skeleton

To ensure human body size constancy, a unit skeleton is defined based on standard body proportions. Standard limb lengths in combination with the directions of the original limbs are used to form the unit skeleton, which can be used to unify the input data for an EHPI3D. The joint positions in this unit skeleton are calculated sequentially starting from the pelvis for each limb as follows:

$$|\vec{d_l}| = \sqrt{(b_x - a_x)^2 + (b_y - a_y)^2 + (b_z - a_z)^2} \tag{4.23}$$

$$\hat{b} = a + \Delta_l |\vec{d_l}| \tag{4.24}$$

where a and b are the joint positions of a limb, d is the normalized direction vector of the particular limb l, $\hat{b}$ is the position of b in the unit skeleton and Δ_l is a constant length based on the limb l represented by a and b. The values of Δ_l are derived from standard body proportions[7]. Figure 4.43 shows the used proportions for the body limbs and Figure 4.44 the proportions for the facial limbs.

In the current case, it is not essential to use exact proportion values, as the same transformation is applied to all inputs. Thus, it is only necessary not to change Δ_l.

[7] `https://de.wikipedia.org/wiki/Körperproportion` (accessed on 2022-03-06)

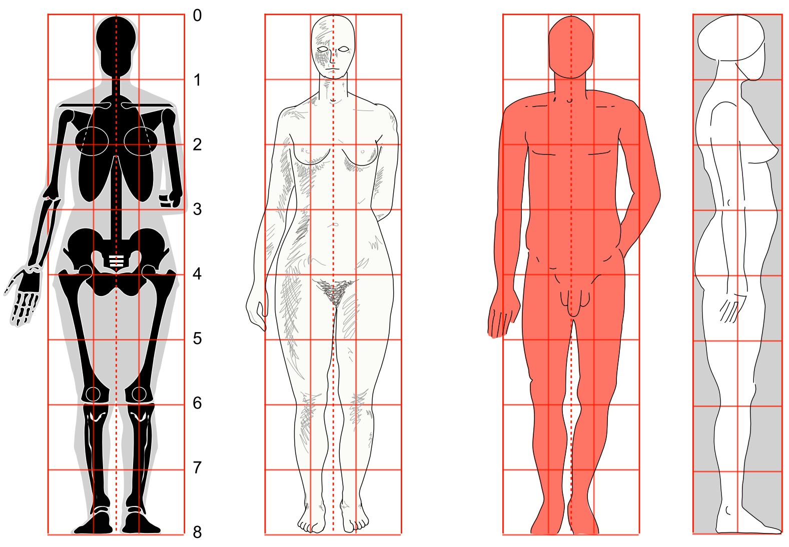


Figure 4.43: Human body proportions. Original from ‘Schnorch’ at Wikipedia[7].

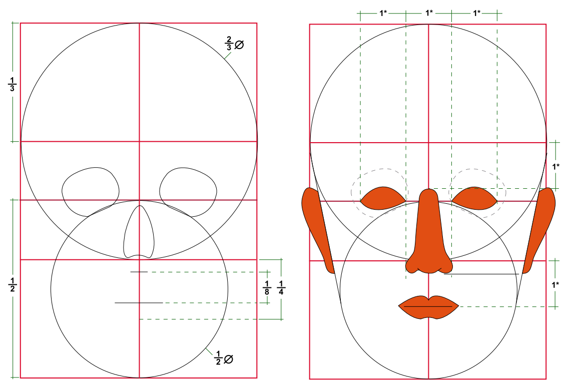


Figure 4.44: Human head proportions. Original from ‘Schnorch’ at Wikipedia[7].

In Figure 4.43, the head size is used as the default unit. The average proportional body size is eight ‘head units’. To specify the joint positions more precisely, these ‘head units’ are separated further into four equally sized parts, resulting in a body size of 32 units. If a human is completely stretched out (arms above the head), the body takes about 38-40 units. Instead of a metric body size, the ‘head units’ are used for the unit skeleton and the skeletal space is normalized to 40 × 40 × 40 ‘head units’. This results in a skeletal representation with fixed limb lengths in a fixed space, and thus a unified representation for an EHPI3D, which only encodes the human motions and no longer individually skeletal features. An overview of all limbs and their length in the unit skeleton is given in Appendix A.4. This work does not contain any investigations regarding the behavior of people with non-standard proportions, such as children.

4.5.3 Training

The ResNet50 architecture is used for the classification of the EHPI3Ds. As a standard multi-label classification problem, the BCE loss function is applied. A standardized training procedure was used for all EHPI3D experiments to keep the results comparable. Due to the small input size of the EHPI3Ds and the lightweight ResNet50 network architecture, the network's training with the full SIM-C01 dataset is finished in under three hours on an NVIDIA RTX 3080.

Optimizer As in the PedRecNet experiments, the AdamW optimizer [139] was used. Furthermore, the learning rate range test presented by Smith [141] was applied to get an initial learning rate based on the steepest gradient. A standard weight decay of $1e^{-02}$ is used. The one-cycle policy [142] was applied. The network was trained with a training cycle of 20 epochs.

Datasets The EHPI experiments mostly used the SIM-C01 training dataset, which contains 17 different actions (see section 4.4.4.3). For some experiments, the real-world datasets used in the previous 2D human pose-based action recognition experiments (see sections 4.2.3 and 4.3.3) were added to gain some real-world examples for the actions 'idle' and 'walk' and some data with the additional actions 'jump', 'sit', 'wave a car out', which are not contained in the SIM-C01 dataset. The data was augmented by flipping the input horizontally in 50% of the cases.

4.5.4 Experiments

First, it is investigated how the different ways to generate an EHPI3D affect the recognition performance on the SIM-C01V dataset. For example, it can vary whether one uses 3D human pose data from ground truth or PedRecNet predictions for input. Going further, the impact of using the unit skeleton is investigated. In addition, the handling of invisible human joints, for example due to occlusion or cropping, in the image must be defined. These joints can either be set to zero or the unreliable estimate of the heatmap output can be used. Further, the impact of including real data from the experiments in sections 4.2 and 4.3 in addition to the simulated training data from the SIM-C01T dataset (A) is investigated. The results of these experiments are shown in Table 4.19.

Since the distribution of actions in the SIM-C01V dataset is unbalanced, the mean balanced accuracy (mBA) is reported in addition to the standard metrics for multi-label classification. The mBA and the F_1-score over the entire dataset (OF_1) are used as the primary metrics. A confidence threshold of 0.65 was used to consider an action as recognized. An overview of all metrics used can be found in section 2.3.1.6. The results show that it is necessary to use 3D human pose data estimated by the PedRecNet already during training in addition to the EHPI3Ds

Experiment	ValS.	mBA	mAP	OF_1	OP	OR	CF_1	CP	CR
P	1	73.2	58.6	79.2	86.3	73.1	52.9	71.3	47.8
G	1	79.6	74.6	85.8	89.7	82.2	67.1	80.9	60.4
G+P	1	81.2	77.3	86.2	91.1	81.9	68.8	81.9	63.4
G+P+E	1	80.6	77.5	85.9	90.9	81.4	68.1	82.9	62.1
G+P+N	1	80.0	76.5	85.9	90.4	81.7	66.7	83.8	61.0
G+P+Z	1	81.6	79.2	86.5	91.3	82.1	70.6	85.0	64.1
G+P (15fps)	1	81.9	80.4	86.7	91.1	82.8	70.4	86.1	64.8
G+P+E (15fps)	1	81.8	80.1	86.2	90.9	82.0	69.8	85.2	64.7
G+P (64frames)	1	83.1	80.4	87.1	91.5	83.1	71.8	84.0	67.1
P	0	75.5	66.0	81.5	87.2	76.4	59.1	76.2	52.3
G	0	69.7	50.3	64.0	67.6	60.7	46.7	60.5	42.9
G+P	0	78.2	71.4	82.6	88.8	77.2	63.7	79.9	57.5
G+P+E	0	77.7	72.3	82.2	88.6	76.7	63.7	82.7	56.5
G+P+N	0	76.1	67.3	81.6	88.1	76.1	60.3	80.1	53.4
G+P+Z	0	78.4	73.2	82.0	88.2	76.6	64.2	81.6	57.9
G+P (15fps)	0	78.4	73.9	82.1	88.0	77.0	63.7	81.9	58.0
G+P+E (15fps)	0	78.4	73.4	82.0	87.8	76.9	64.2	81.2	58.1
G+P (64frames)	0	81.0	78.8	83.9	89.4	79.1	69.6	84.9	63.1

Table 4.19: EHPI3D action recognition results on the SIM-C01V dataset. The influence of different kinds of input pose data and preprocessing on the performance are compared. Legend: G = 3D human pose input from ground truth, P = 3D human pose input from PedRecNet predictions, N = no unit skeleton was used, E = additionally used real-world data from the experiments shown in section 4.2 and 4.3 as training data, and Z = zeroed out invisible joints. In G+P experiments 65% ground truth human pose data and 35% PedRecNet predicted human pose data were used. ValS. displays whether human pose data from ground truth (G) or PedRecNet predictions (P) were used as EHPI3D input during validation. (15fps) stands for 32 frames with 15fps input = 2.12 seconds input and (64frames) means a double width EHPI3D with encoding also 2.12 seconds but with 30fps.

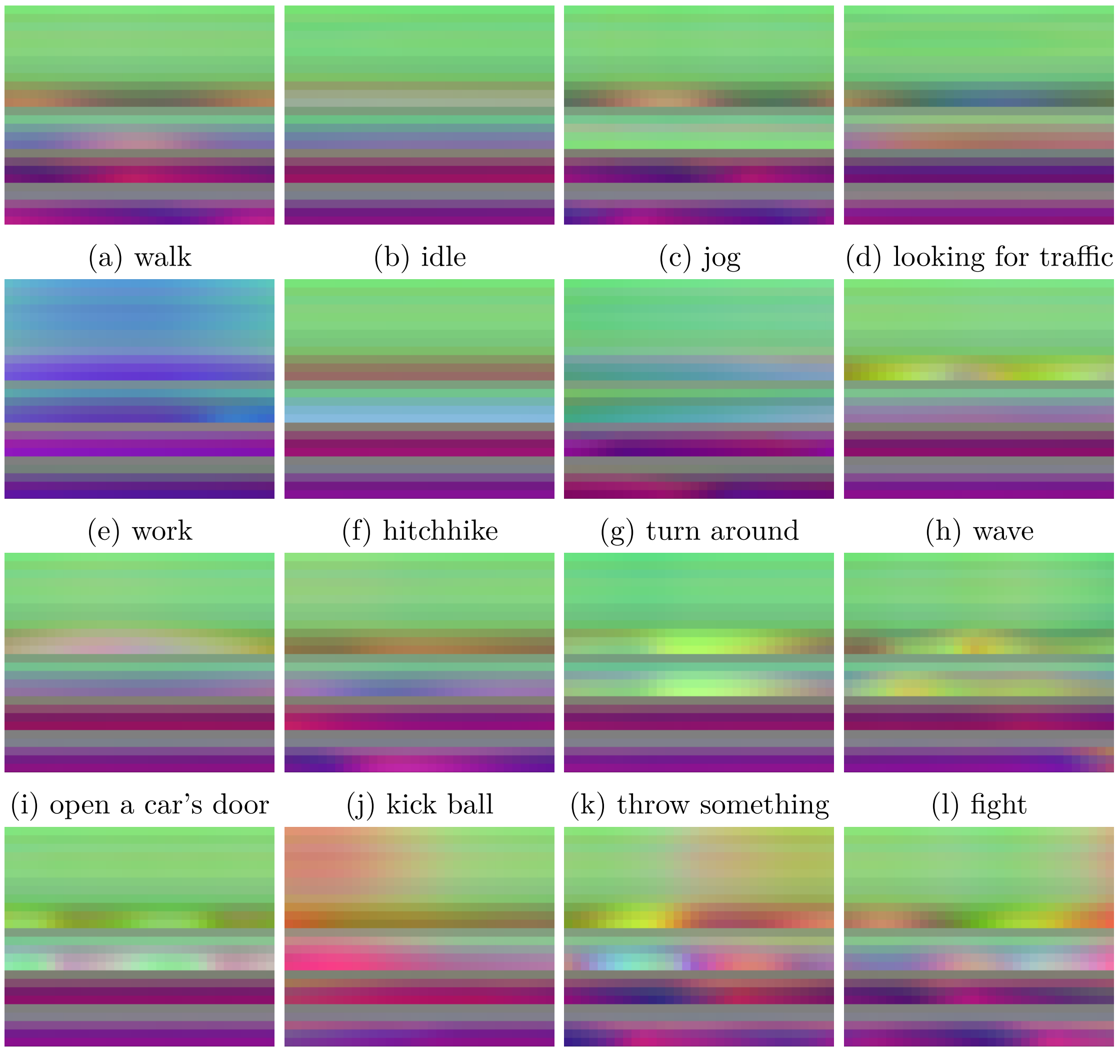
generated with ground truth 3D human pose data. If only ground truth data is used, the F_1 score is 24.1%, which is worse than using additional estimated 3D human pose data. This behavior was to be expected, since under real conditions, there is always some noise in the joint coordinates due to the 3D human pose prediction, which should be considered in the training. The usage of 100% estimated 3D human pose data leads to good results, but better results can be reached by combining it with ground truth 3D human pose input data. In the experiments, a mixture of 65% ground truth human pose data to 35% estimated human pose data proved to be a good compromise. The additional use of real training data from the previous experiments slightly decreases the performance, hinting that the action execution might differ from the executions in the SIM-C01 dataset. However, it must be noted here that the real data only includes examples of 'idle' and 'walk' which also appear in the SIM-C01 dataset. Using the unit skeleton, and thus removing skeletal size dependence in an EHPI3D, shows an improvement of 2.8% in mBA and 1.3% in OF_1-score. In contrast, zeroing out unseen human joints slightly worsens the results. The performance can be improved further by using a larger time frame of 64 frames on 30 fps input, yet at the cost of potentially longer reaction times until an action is recognized. Using 15fps results in slightly worse results, hinting that some actions may be too subtle to be recognized with a lower frame rate. Overall, with an OF_1 score of 82.6% and mBA of 78.2%, good results are achieved with the G+P approach. Those results are broken down in detail per action in Table 4.20.

The basic movements 'stand', 'walk', and 'jog' have reasonable recognition rates. Some more apparent actions, like 'hitchhiking' or 'looking for traffic', are also well recognized. For other actions, however, the recognition is significantly worse. 'Stand up', for example, only achieves an F_1 score of 39.6%, which can be explained by the missing context of the environment and especially of another person. An additional factor is the composition of the action 'fight' which has several overlaps with other actions, for example, 'stand' and 'stumble'. For actions such as 'throw something', 'kick a ball', or 'open a car's door', further contextual information would also be useful. The number of frames per action varies greatly, since some actions, such as 'fall', only occur in a few frames during a scene. Other actions such as 'walk' or 'stand', on the other hand, are included in almost all scenes, and there is a correspondingly large amount of data for them. In a new dataset, care should be taken to record some underrepresented actions separately in addition to the overall scenes to obtain a somewhat larger training and validation basis. For the interaction of several actions in real scenes, however, it is still relevant to record entire motion sequences. An overview of the representation of the individual actions is also provided by Figure 4.45 which shows an example EHPI3D for each action.

In Appendix C.1 there is another graphic where in addition to the EHPI3Ds, the human in the last frame of the EHPI3D is also shown. The 'idle' (4.45b) examples show the static nature without major movements. The examples of 'walking' (4.45a)

Action	BA	AP	F_1	Precision	Recall	N
stand	90.0	95.3	88.2	93.3	83.6	45652
idle	64.2	59.7	42.7	81.2	29.0	10168
walk	90.2	95.4	89.6	90.5	88.8	57699
jog	86.0	87.4	79.1	85.1	73.9	16207
wave	87.5	85.8	77.7	80.0	75.6	2970
kick a ball	81.5	77.4	71.9	83.6	63.0	1144
throw something	69.5	59.2	55.0	93.0	39.1	1024
looking for traffic	88.7	89.2	82.3	85.7	79.1	14819
hitchhike	91.2	88.8	82.1	81.0	83.1	4018
turn around	75.1	66.1	60.9	76.2	50.7	3494
work	75.1	81.7	65.0	91.3	50.4	4263
argue	89.4	78.3	76.7	74.4	79.1	961
stumble	65.3	34.8	41.5	64.1	30.7	837
open a car’s door	65.6	66.1	46.8	92.6	31.3	1192
fall	87.7	58.7	55.6	43.9	76.0	591
stand up	58.7	50.4	27.8	69.4	17.4	873
fight	63.5	40.4	39.6	72.7	27.2	990

Table 4.20: EHPI3D action recognition results on the SIM-C01V dataset per action. The results are from the G+P+E experiment and ordered by the number of samples (N) per action.



(a) walk (b) idle (c) jog (d) looking for traffic

(e) work (f) hitchhike (g) turn around (h) wave

(i) open a car's door (j) kick ball (k) throw something (l) fight

(m) argue (n) stand up (o) stumble (p) fall

Figure 4.45: EHPI3D examples. Stand is not contained as a separate action, as it would equal idling.

and ‘jogging’ (4.45c) both show a sinusoidal motion pattern of the two legs, but differ in their frequency. In the example ‘hitchhike’ (4.45f) one can see clearly the color shift of the right arm, which is lifted. Likewise, in the example ‘turn around’ (4.45g) one can see how the colors shift in time due to a rotation of the human. Something similar can be observed in the ‘stand up’ (4.45n) example, where the colors shift in the head and upper body area. In the ‘wave’ (4.45h), ‘fight’ (4.45l), ‘throw something’ (4.45k), and ‘argue’ (4.45m) examples, movements of the two arm parts are mainly visible. Interesting are also the examples ‘stumble’ (4.45o) and ‘fall’ (4.45p), in which abrupt accelerations are to be recognized, which is atypical opposite the normally smoother movement sequences. The action recognition should be able to distinguish these actions from standard movements. However, since these two actions are among the actions with by far the least sample data, it can be assumed that more data could significantly improve the overall result for detecting people stumbling or falling.

4.5.5 Discussion

This work introduced an elaborate action recognition dataset that includes important pedestrian-related actions as multi-label classification data. On the novel simulated pedestrian action dataset it was shown that recognition of basic motion types such as standing, walking, and jogging is possible with an OF_1 score above 80%. Many other actions are also well recognized, but the usage of only 3D human pose data as input to the action recognition becomes problematic for some actions. For the recognition of, for example, fighting, kicking a ball, or throwing something, additional context information would be beneficial. In subjective evaluations on real data, the action recognition shows promising results, especially in basic movement types. However, a quantitative analysis based on a real dataset is future work. Through the simulation approach, it could be investigated how the classification behavior of the EHPI approach extended to 3D human pose recognition.

4.6 Conclusions from all experiments

Through the applications of simulated data shown in this chapter, the following hypotheses were to be investigated:

1. The use of simulation data for training deep neural networks for vision tasks, which receive purely visual sensor information as input, leads to overfitting on simulation data.

2. By using abstraction layers, such as human joint positions, instead of direct features from the visual input, action recognition systems can be trained with simulation data only.

3. Labels, that are not currently present in real datasets, can be created with the simulation framework and used to train DNNs to achieve functionality which would not be possible with current, freely available datasets.

In the corner case experiments in section 4.1, hypothesis one was confirmed. The recognition rate of corner cases by a 2D human pose recognition algorithm was improved through targeted simulation. However, the experiments showed that overfitting occurs due to an excessive amount of simulation data, and performance on real data decreases. Training purely on simulation data and applying it to real data is not possible with the demonstrated methods.

Hypothesis two was investigated in the experiments in section 4.3 and section 4.4 using examples from action recognition as well as head orientation estimation. It was shown that by using human pose estimation data as an abstraction layer, it is possible to train systems purely on simulation data and that the results apply to real data. Hypothesis two can thus also be confirmed.

Sections 4.4 and 4.5, showed how the motion capture manager and simulation environment allowed us to generate simulation data with labels for body and head orientation estimation as well as special pedestrian-related action recognition labels that do not exist in this way in any real datasets. The systems perform well on a simulated validation dataset. Accordingly, hypothesis three can also be confirmed. In first subjective tests, the systems trained with this data could be used for real data. Objective tests on real data are currently not possible due to missing datasets.

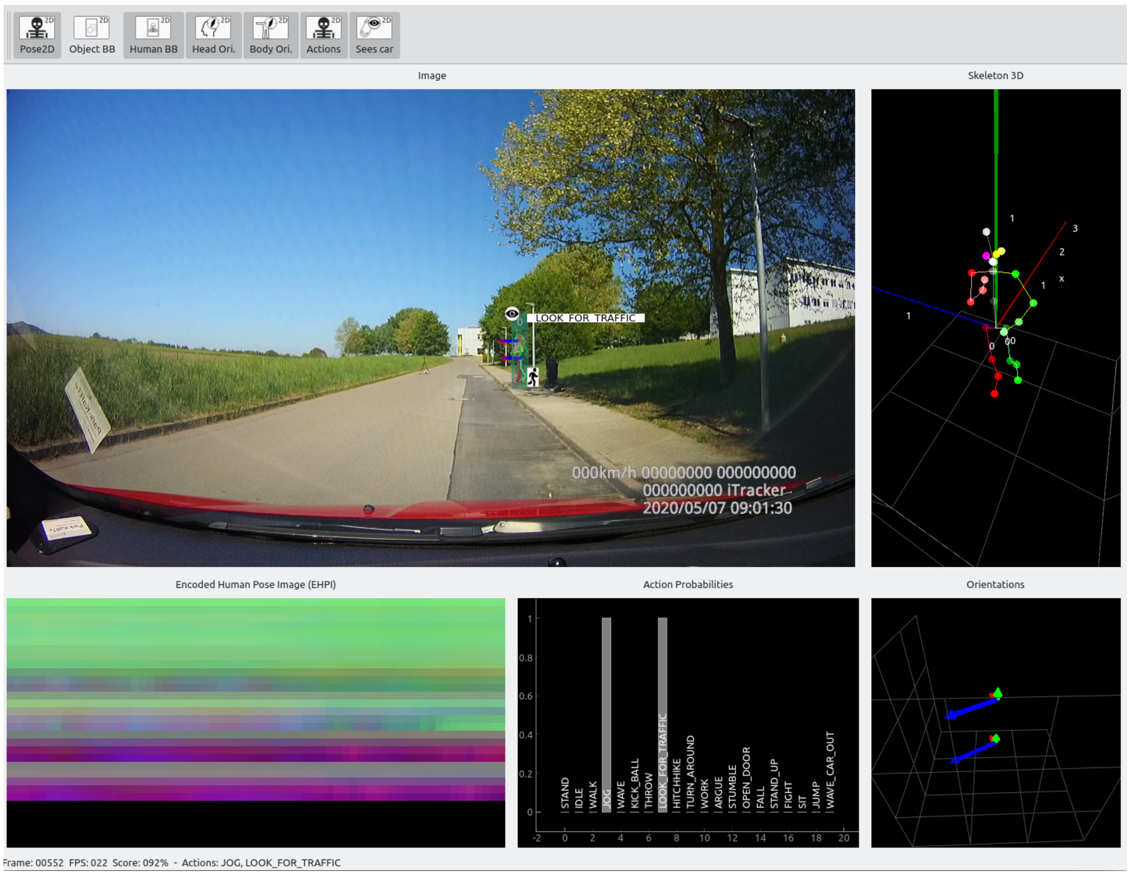
Chapter 5

Conclusion

This work shows how simulation data can be used in many ways to improve deep learning applications, validate them, or enable applications that would otherwise not be possible due to a lack of data. With the demonstrated motion capture management system, it is possible to efficiently record motion data in a motion capture lab, annotate it, and integrate it into the presented simulation environment. The simulation framework makes it possible to generate a huge amount of data from different persons, movements, environments, and situations without much manual effort and to generate training data for machine learning methods. It has been shown that this allows 2D human pose estimation methods to be adapted to detect corner-case motions. Further experiments have shown, that 2D human pose-based action recognition can be trained with simulated data, and that the results can also be transferred to real data. This procedure was continued and resulted in the development of a 3D human pose-based action recognition approach (EHPI3D). An extensive action catalog with pedestrian-related actions was generated. The recorded motions were used in a simulation dataset that contains specific pedestrian actions and even supported multiple actions simultaneously (multi-label). The EHPI3D human pose recognition approach can predict these actions with high accuracy. However, for even better accuracy in the future, the context has to be included in the action recognition and the pose information. Furthermore, in the context of this work, the PedRecNet was developed, which can simultaneously perform 3D human pose recognition as well as head and body orientation estimation in addition to 2D human pose recognition. The network can perform these recognition tasks on a consumer GPU at 15-30 frames per second, including action recognition, depending on the number of people. In the experiments, the PedRecNet could be trained stably and without significant effort. The individual tasks of the network achieve competitive performance against specialized SOTA methods, though not outperforming them. It is very well suited as a baseline for diverse work. The network supports a correspondingly large number of tasks in the field of human recognition, yet the training procedure is stable and straightforward. Furthermore, it achieves competitive performance compared to other SOTA approaches in the respective tasks.

In conclusion, simulation data, especially when not working directly on visual input data, is well suited to train and validate machine learning methods. However, it is usually advisable to additionally use real data, especially to train the parts of a neural network that are directly based on visual input. To work directly on visual data, it appears to be helpful to investigate methods in the field of domain adaptation to further reduce the gap between simulation and real data on visual sensor data.

Finally, the developed system is to be reviewed against possible open questions in the field of pedestrian recognition shown in Figure 1.1 in the introduction. Figure 5.1 shows an exemplary integration of all technologies and an execution on real data, recorded with the same action catalog used for the SIM-C01 dataset.



Frame: 00552 FPS: 022 Score: 092% - Actions: JOG, LOOK_FOR_TRAFFIC

Figure 5.1: PedRec application with all available information recognized by the PedRecNet (2D- and 3D human pose estimation, head and body orientation estimation) and EHPI3D (action recognition). The eye symbol indicates that the person is looking towards the observing sensor platform. This function is based on the head orientation estimation.

The figure shows that in addition to object detection and 2D and 3D human pose estimation, head and body orientation estimation, and, based on these, an estimation of whether one's vehicle is within the pedestrian's field of view were integrated. Labels for these tasks hardly exist in conventional datasets, but thanks to the simulation, they can be generated quickly and easily. In addition, the action recognition output, based on the EHPI3D network, is visualized. It shows good results on this real data example and contains labels that do not exist in other datasets but provide valuable inputs for further functions in the context of pedestrian recognition.

Appendix A

Tables

A.1 Reutlingen University - Motion Capture Lab: Hardware Overview

Reutlingen University - Motion Capture Lab: Hardware Overview. These cameras were utilized for recordings used in this work.

Camera	Type	Quantity	Resolution (MP)	Max Frame Rate (Hz)	Focal Length (mm)
Vantage V8	IR	8	8	260	12.5
Vantage V5	IR	12	5	420	12.5
Vue	RGB	2	2.1	120	6 – 12

A.2 SIM-C01: Record Catalogue

The sequences where varied in using different accessories and actions (see Appendix A.3) in either the left hand, right hand or both hands, movements (*stand*, *walk* or *jog*) and movement speeds (*slow*, *normal*, *none*).

Sequence name	Num. Records
Cross the street between cars	11
Cross the street between cars and back up	11
Move (left to right)	77

continues on next page

Sequence name	Num. Records
Move (right to left)	77
Cross the street on a crosswalk and carefully look (right sidewalk)	66
Carry a shopping bag (right sidewalk)	22
Cross the street on a crosswalk and no look (right sidewalk)	20
Wave to someone on the opposite sidewalk (left sidewalk)	11
Hitchhike (left sidewalk and walking)	11
Yell at someone and argue	11
Wave car in	11
Load a vehicle (right sidewalk - automatic)	11
Move (left to right, jogging)	55
Cross the street on a crosswalk and no look (left sidewalk)	20
Throw a ball	22
Carry a shopping bag (left sidewalk)	33
Cross the street on a crosswalk and carefully look (left sidewalk)	66
Hitchhike (right sidewalk and walking)	11
Wait for somebody on a crosswalk (left sidewalk)	22
Cross the street on a crosswalk and quick look (left sidewalk)	66
Hitchhike (standing and right sidewalk)	11
Move (right to left, jogging)	55
Wave to someone on the same sidewalk (right sidewalk)	11
Hitchhike (standing and left sidewalk)	11
Getting called by someone behind (right sidewalk)	11
Get called by someone on the opposite sidewalk (right sidewalk)	11
Move and fall	22
Cross the street on a crosswalk and quick look (right sidewalk)	66
Fight with someone	11
Load a vehicle (left sidewalk - manual)	11
Lean against a wall waiting	22
Cross the street on a crosswalk and carefully look and turn around (right sidewalk)	66
Wave to someone on the same sidewalk (left sidewalk)	11
Move and stumble	22
Cross the street on a crosswalk and carefully look and turn around (left sidewalk)	66
Wait for somebody on a crosswalk (right sidewalk)	22
Load a vehicle (right sidewalk - manual)	11
Wave to someone on the opposite sidewalk (right sidewalk)	11

continues on next page

Sequence name	Num. Records
Open a car door (right sidewalk)	11
Kick a ball	22
Open a car door (left sidewalk)	11
Load a vehicle (left sidewalk - automatic)	11

A.3 SIM-C01: Used Accessories

Accessory	Possible Actions
Smartphone	Phone, Text
Stroller	Push
Crate	Carry, Load, Unload
Shopping bag	Carry
Ball	Kick, Throw, Catch
Rolling suitcase	Pull
Umbrella	Hold
Walking stick	Use

A.4 EHPI3D: Unit Skeleton

Overview of all limbs and their length in the unit skeleton used in our EHPI3D experiments. Limbs are represented by their starting joint l_a and end joint l_b.

Limb (joint l_a)	Limb (joint l_b)	Limb length Δ_l [‘head units’] (see 4.5.2)
pelvis	left hip	2
pelvis	right hip	2
pelvis	spine center	5
right hip	right knee	8
right knee	right ankle	8
left hip	left knee	8
left knee	left ankle	8
left ankle	left foot end	4
right ankle	right foot end	4
spine center	neck	6
neck	head lower	1
head lower	head upper	4
neck	left shoulder	3
neck	right shoulder	3
right shoulder	right elbow	6
right elbow	right wrist	5
left shoulder	left elbow	6
left elbow	left wrist	5
left wrist	left hand end	3
right wrist	right hand end	3
head lower	nose	2.25
nose	right eye	1
right eye	right ear	1.25
nose	left eye	1
left eye	left ear	1.25

Appendix B

Listings

B.1 Character Description File

Example of a character description file which contains meta information about a 3D human avatar.

```
{
  "age": 31,
  "size": 187,
  "sizeIsEstimated": false,
  "uid": "42721607-8067-4247-8cb0-fe864368ad2e",
  "weight": 80,
  "weightIsEstimated": false,
  "gender": "Male",
  "skinColor": "(III) Medium, white to light brown"
}
```

B.2 Motion Capture Description File

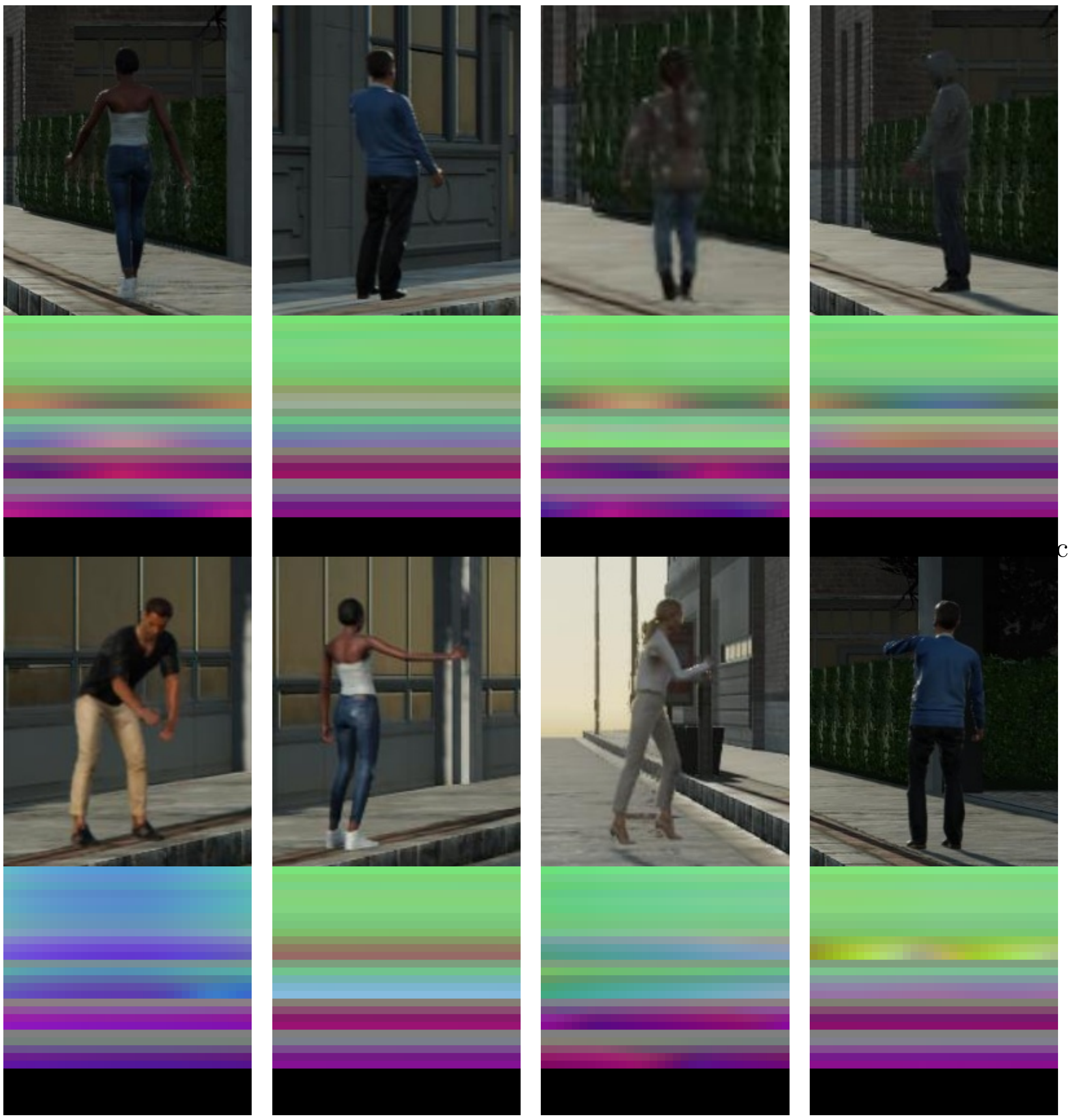
Example of a motion description file which contains meta information about a motion capture record.

```
{
  "createdAt": "2019-10-01T11:15:39.625Z",
  "dir": "2020/01/15/1",
  "humanBodyParts": [],
  "humanConfiguration": "Normal",
  "id": "1",
  "instructions": [
    {
      "description": "Walking",
      "location": "Left sidewalk"
    },
    {
      "description": "Hitchhike",
      "location": "Left sidewalk"
    }
  ],
  "movement": "Walking",
  "movementSpeed": "Normal",
  "participant": {
    "age": 30,
    "size": 185,
    "sizeIsEstimated": true,
    "uid": "cd4dc166-51ea-41ea-88c5-2c1e65225d43",
    "weight": 93,
    "weightIsEstimated": true,
    "gender": "Male",
    "skinColor": "(II) White, fair"
  },
  "project": "C. Katalog #1",
  "scene": "Straight street with two sidewalks",
  "sequence": "Hitchhiking (left sidewalk, walking)",
  "actions": [
    {
      "frame": 63,
      "actions": [
        "Move"
      ]
    },
    {
      "frame": 272,
      "actions": [
        "Looking for traffic",
        "Move"
      ]
    },
    {
      "frame": 400,
      "actions": [
        "Hitchhike",
        "Move"
      ]
    },
    {
      "frame": 791,
      "actions": [
        "Hitchhike"
      ]
    },
    {
      "frame": 1001,
      "actions": [
        "Stop"
      ]
    }
  ]
}
```

Appendix C

Figures

C.1 EHPI3D Examples



c

Figure C.1: EHPI3D Examples (A)

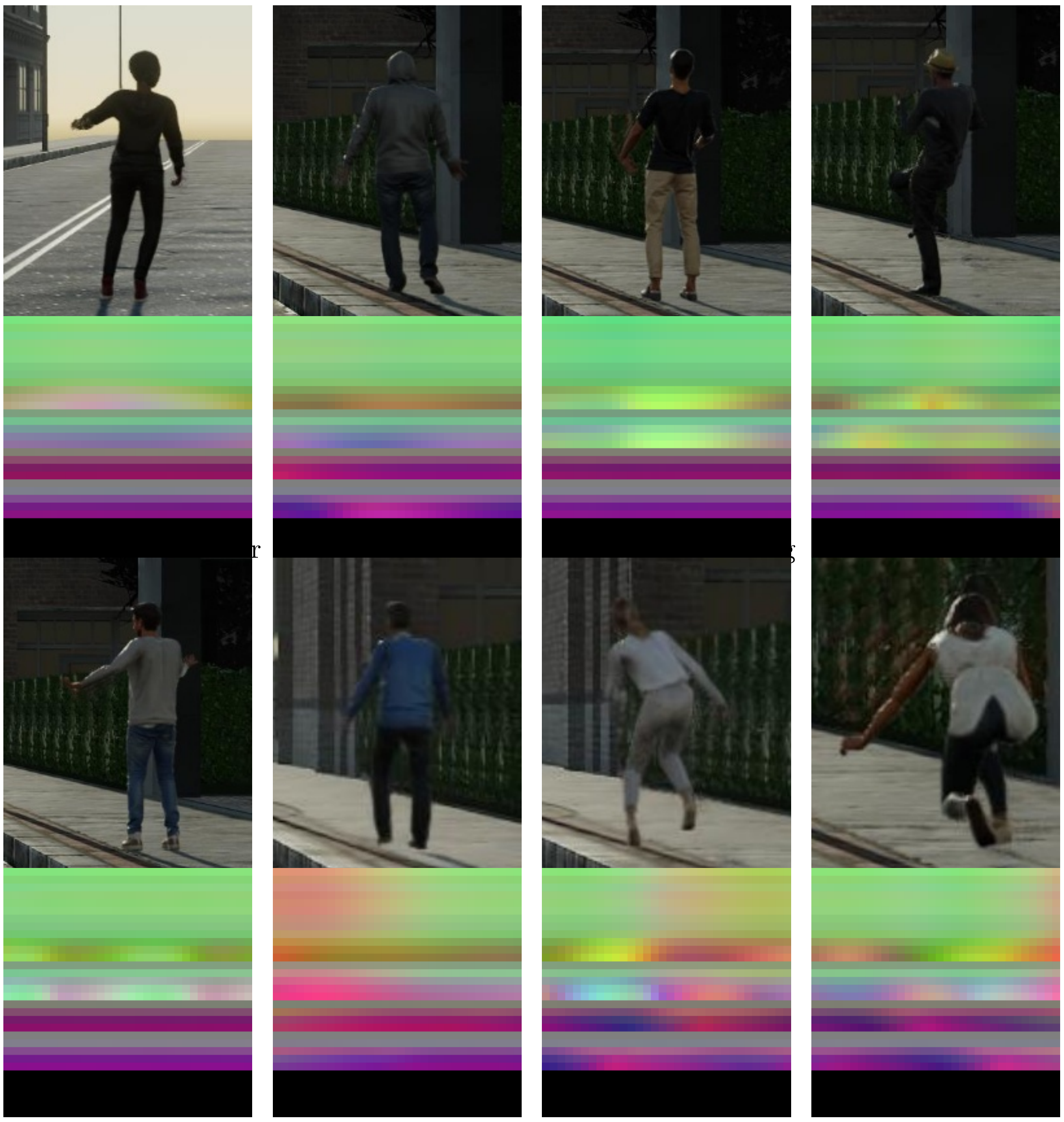


Figure C.2: EHPI3D Examples (B)

Bibliography

[1] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. Cambridge, Massachusetts: The MIT Press, Jan. 2017.

[2] R. Brooks, 'The Big Problem With Self-Driving Cars Is People', *IEEE Spectrum: Technology, Engineering, and Science News*, Aug. 2017.

[3] C. Northcutt, L. Jiang and I. Chuang, 'Confident Learning: Estimating Uncertainty in Dataset Labels', *Journal of Artificial Intelligence Research*, vol. 70, pp. 1373–1411, Apr. 2021.

[4] C. G. Northcutt, A. Athalye and J. Mueller, 'Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks', in *Thirty-Fifth Conference on Neural Information Processing Systems (NeurIPS)*, Apr. 2021.

[5] D. Ludl, T. Gulde, S. Thalji and C. Curio, 'Using Simulation to Improve Human Pose Estimation for Corner Cases', in *21st IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 3575–3582.

[6] D. Ludl, T. Gulde and C. Curio, 'Simple yet efficient real-time pose-based action recognition', in *22nd IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)*, Nov. 2019, pp. 581–588.

[7] D. Ludl, T. Gulde and C. Curio, 'Enhancing Data-Driven Algorithms for Human Pose Estimation and Action Recognition Through Simulation', *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2020.

[8] D. Burgermeister and C. Curio, 'PedRecNet: Multi-task deep neural network for full 3D human pose and orientation estimation', Apr. 2022.

[9] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'ImageNet classification with deep convolutional neural networks', in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Red Hook, NY, USA: Curran Associates Inc., Dec. 2012, pp. 1097–1105.

[10] K. Simonyan and A. Zisserman, 'Very Deep Convolutional Networks for Large-Scale Image Recognition', in *3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015.

[11] K. He, X. Zhang, S. Ren and J. Sun, 'Deep Residual Learning for Image Recognition', in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Dec. 2015, pp. 770–778.

[12] O. Russakovsky *et al.*, 'ImageNet Large Scale Visual Recognition Challenge', *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, Jan. 2015.

[13] M. Tan and Q. V. Le, 'EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks', *arXiv:1905.11946 [cs, stat]*, Sep. 2020.

[14] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel and A. Vaswani, 'Bottleneck Transformers for Visual Recognition', *arXiv:2101.11605 [cs]*, Jan. 2021.

[15] A. Krizhevsky, 'Learning Multiple Layers of Features from Tiny Images', p. 60, 2009.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and F.-F. Li, 'ImageNet: A large-scale hierarchical image database', in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2009, pp. 248–255.

[17] I. Krasin *et al.*, 'OpenImages: A public dataset for large-scale multi-label and multi-class image classification.', *Dataset available from https://github.com/openimages*, 2017.

[18] R. Girshick, J. Donahue, T. Darrell and J. Malik, 'Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation', in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587.

[19] R. Girshick, 'Fast R-CNN', in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1440–1448.

[20] S. Ren, K. He, R. Girshick and J. Sun, 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks', *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[21] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, 'You Only Look Once: Unified, Real-Time Object Detection', in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 779–788.

[22] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, 'YOLOv4: Optimal Speed and Accuracy of Object Detection', *arXiv:2004.10934 [cs, eess]*, Apr. 2020.

[23] A. Geiger, 'Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite', in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '12, Washington, DC, USA: IEEE Computer Society, 2012, pp. 3354–3361.

[24] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn and A. Zisserman, ‘The Pascal Visual Object Classes (VOC) Challenge’, *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[25] T.-Y. Lin *et al.*, ‘Microsoft COCO: Common Objects in Context’, in *European Conference on Computer Vision (ECCV)*, Springer International Publishing, 2014, pp. 740–755.

[26] L. Pishchulin *et al.*, ‘DeepCut: Joint Subset Partition and Labeling for Multi Person Pose Estimation’, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 4929–4937.

[27] Z. Cao, T. Simon, S.-E. Wei and Y. Sheikh, ‘Realtime multi-person 2d pose estimation using part affinity fields’, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[28] A. Newell, K. Yang and J. Deng, ‘Stacked Hourglass Networks for Human Pose Estimation’, in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe and M. Welling, Eds., vol. 9912, Cham: Springer International Publishing, 2016, pp. 483–499.

[29] U. Iqbal and J. Gall, ‘Multi-person Pose Estimation with Local Joint-to-Person Associations’, in *Computer Vision – ECCV 2016 Workshops*, G. Hua and H. Jégou, Eds., vol. 9914, Cham: Springer International Publishing, 2016, pp. 627–642.

[30] B. Xiao, H. Wu and Y. Wei, ‘Simple Baselines for Human Pose Estimation and Tracking’, in *European Conference on Computer Vision (ECCV)*, Springer International Publishing, 2018, pp. 472–487.

[31] S. Johnson and M. Everingham, ‘Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation’, in *Procedings of the British Machine Vision Conference 2010*, Aberystwyth: British Machine Vision Association, 2010, pp. 12.1–12.11.

[32] B. Sapp and B. Taskar, ‘MODEC: Multimodal Decomposable Models for Human Pose Estimation’, in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, USA: IEEE, Jun. 2013, pp. 3674–3681.

[33] M. Andriluka, L. Pishchulin, P. Gehler and B. Schiele, ‘2D Human Pose Estimation: New Benchmark and State of the Art Analysis’, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[34] D. Mehta *et al.*, ‘VNect: Real-time 3D human pose estimation with a single RGB camera’, *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–14, Jul. 2017.

[35] D. Mehta *et al.*, ‘Single-Shot Multi-person 3D Pose Estimation from Monocular RGB’, in *2018 International Conference on 3D Vision (3DV)*, Verona: IEEE, Sep. 2018, pp. 120–130.

[36] D. Mehta *et al.*, ‘XNect: Real-time multi-person 3D motion capture with a single RGB camera’, *ACM Transactions on Graphics*, vol. 39, no. 4, Jul. 2020.

[37] D. C. Luvizon, H. Tabia and D. Picard, ‘Multi-task Deep Learning for Real-Time 3D Human Pose Estimation and Action Recognition’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[38] C.-H. Chen and D. Ramanan, ‘3D Human Pose Estimation = 2D Pose Estimation + Matching’, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 5759–5767.

[39] D. Pavllo, C. Feichtenhofer, D. Grangier and M. Auli, ‘3D human pose estimation in video with temporal convolutions and semi-supervised training’, in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[40] B. Tekin, P. Marquez-Neila, M. Salzmann and P. Fua, ‘Learning to Fuse 2D and 3D Image Cues for Monocular Body Pose Estimation’, in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice: IEEE, Oct. 2017, pp. 3961–3970.

[41] X. Zhou, Q. Huang, X. Sun, X. Xue and Y. Wei, ‘Towards 3D Human Pose Estimation in the Wild: A Weakly-Supervised Approach’, in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice: IEEE, Oct. 2017, pp. 398–407.

[42] N. Kolotouros, G. Pavlakos, M. Black and K. Daniilidis, ‘Learning to Reconstruct 3D Human Pose and Shape via Model-Fitting in the Loop’, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South): IEEE, Oct. 2019, pp. 2252–2261.

[43] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero and M. J. Black, ‘Keep It SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image’, in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe and M. Welling, Eds., vol. 9909, Cham: Springer International Publishing, 2016, pp. 561–578.

[44] X. Zhou, X. Sun, W. Zhang, S. Liang and Y. Wei, ‘Deep Kinematic Pose Regression’, in *Computer Vision – ECCV 2016 Workshops*, G. Hua and H. Jégou, Eds., vol. 9915, Cham: Springer International Publishing, 2016, pp. 186–201.

[45] D. C. Luvizon, D. Picard and H. Tabia, '2D/3D Pose Estimation and Action Recognition Using Multitask Deep Learning', in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA: IEEE, Jun. 2018, pp. 5137–5146.

[46] J. Martinez, R. Hossain, J. Romero and J. J. Little, 'A Simple Yet Effective Baseline for 3d Human Pose Estimation', in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice: IEEE, Oct. 2017, pp. 2659–2668.

[47] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll and M. J. Black, 'SMPL: A Skinned Multi-person Linear Model', *ACM Trans. Graph.*, vol. 34, no. 6, pp. 1–16, Oct. 2015.

[48] C. Ionescu, P. Papava, V. Olaru and C. Sminchisescu, 'Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, Jul. 2014.

[49] L. Sigal, A. O. Balan and M. J. Black, 'HumanEva: Synchronized Video and Motion Capture Dataset and Baseline Algorithm for Evaluation of Articulated Human Motion', *International Journal of Computer Vision*, vol. 87, no. 1-2, pp. 4–27, Mar. 2010.

[50] D. Mehta *et al.*, 'Monocular 3D Human Pose Estimation in the Wild Using Improved CNN Supervision', in *2017 International Conference on 3D Vision (3DV)*, Qingdao: IEEE, Oct. 2017, pp. 506–516.

[51] T. von Marcard, R. Henschel, M. J. Black, B. Rosenhahn and G. Pons-Moll, 'Recovering Accurate 3D Human Pose in the Wild Using IMUs and a Moving Camera', in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu and Y. Weiss, Eds., vol. 11214, Cham: Springer International Publishing, 2018, pp. 614–631.

[52] M. Andriluka, S. Roth and B. Schiele, 'Pictorial structures revisited: People detection and articulated pose estimation', in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 1014–1021.

[53] M. Enzweiler and D. M. Gavrila, 'Integrated pedestrian classification and orientation estimation', in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, USA: IEEE, Jun. 2010, pp. 982–989.

[54] F. Flohr, M. Dumitru-Guzu, J. F. P. Kooij and D. M. Gavrila, 'Joint probabilistic pedestrian head and body orientation estimation', in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, MI, USA: IEEE, Jun. 2014, pp. 617–622.

[55] F. Flohr, M. Dumitru-Guzu, J. F. P. Kooij and D. M. Gavrila, ‘A Probabilistic Framework for Joint Pedestrian Head and Body Orientation Estimation’, *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1872–1882, Aug. 2015.

[56] A. Gupta, K. Thakkar, V. Gandhi and P. J. Narayanan, ‘Nose, Eyes and Ears: Head Pose Estimation by Locating Facial Keypoints’, in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, United Kingdom: IEEE, May 2019, pp. 1977–1981.

[57] N. Ruiz, E. Chong and J. M. Rehg, ‘Fine-Grained Head Pose Estimation Without Keypoints’, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Salt Lake City, UT, USA: IEEE, Jun. 2018, pp. 2155–215 509.

[58] L. Pan, S. Ai, Y. Ren and Z. Xu, ‘Self-Paced Deep Regression Forests with Consideration on Underrepresented Examples’, in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox and J.-M. Frahm, Eds., vol. 12375, Cham: Springer International Publishing, 2020, pp. 271–287.

[59] Z. Hu, Y. Xing, C. Lv, P. Hang and J. Liu, ‘Deep convolutional neural network-based Bernoulli heatmap for head pose estimation’, *Neurocomputing*, vol. 436, pp. 198–209, May 2021.

[60] R. Valle, J. M. Buenaposada and L. Baumela, ‘Multi-task head pose estimation in-the-wild’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[61] J. Xia, H. Zhang, S. Wen, S. Yang and M. Xu, ‘An Efficient Multitask Neural Network for Face Alignment, Head Pose Estimation and Face Tracking’, *arXiv:2103.07615 [cs]*, Mar. 2021.

[62] D. Heo, J. Nam and B. Ko, ‘Estimation of Pedestrian Pose Orientation Using Soft Target Training Based on Teacher–Student Framework’, *Sensors*, vol. 19, no. 5, p. 1147, Mar. 2019.

[63] D. Lee, M.-H. Yang and S. Oh, ‘Head and Body Orientation Estimation Using Convolutional Random Projection Forests’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 1, pp. 107–120, Jan. 2019.

[64] M. Steinhoff and D. Göhring, ‘Pedestrian Head and Body Pose Estimation with CNN in the Context of Automated Driving:’ in *Proceedings of the 6th International Conference on Vehicle Technology and Intelligent Transport Systems*, Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2020, pp. 353–360.

[65] C. Wu *et al.*, 'MEBOW: Monocular Estimation of Body Orientation in the Wild', in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 3448–3458.

[66] D. Tosato, M. Spera, M. Cristani and V. Murino, 'Characterizing Humans on Riemannian Manifolds', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1972–1984, Aug. 2013.

[67] M. Andriluka, S. Roth and B. Schiele, 'Monocular 3D pose estimation and tracking by detection', in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, USA: IEEE, Jun. 2010, pp. 623–630.

[68] Y. Huang, S.-H. Lai and S.-H. Tai, 'Human Action Recognition Based on Temporal Pose CNN and Multi-dimensional Fusion', in *European Conference on Computer Vision (ECCV) Workshops*, Springer International Publishing, 2018, pp. 426–440.

[69] Z. Tu *et al.*, 'Multi-stream CNN: Learning representations based on human-related regions for action recognition', *Pattern Recognition*, vol. 79, pp. 32–43, Jul. 2018.

[70] M. Zolfaghari, G. L. Oliveira, N. Sedaghat and T. Brox, 'Chained Multi-stream Networks Exploiting Pose, Motion, and Appearance for Action Classification and Detection', in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[71] W. Du, Y. Wang and Y. Qiao, 'RPAN: An End-to-End Recurrent Pose-Attention Network for Action Recognition in Videos', in *IEEE Int. Conf. on Computer Vision (ICCV)*, Oct. 2017, pp. 3745–3754.

[72] S. Song, C. Lan, J. Xing, W. Zeng and J. Liu, 'An End-to-End Spatio-Temporal Attention Model for Human Action Recognition from Skeleton Data', *AAAI Conference on Artificial Intelligence*, pp. 4263–4270, 2017.

[73] Z. Fang, D. Vázquez and A. López, 'On-Board Detection of Pedestrian Intentions', *Sensors*, vol. 17, p. 2193, Sep. 2017.

[74] M. Garbade and J. Gall, 'Handcrafting vs Deep Learning: An Evaluation of NTraj+ Features for Pose Based Action Recognition', in *Workshop: New Challenges in Neural Computation (NC2)*, Sep. 2016.

[75] S. Laraba, M. Brahimi, J. Tilmanne and T. Dutoit, '3D skeleton-based action recognition by representing motion capture sequences as 2D-RGB images', *Computer Animation and Virtual Worlds*, vol. 28, no. 3-4, e1782, 2017.

[76] V. Choutas, P. Weinzaepfel, J. Revaud and C. Schmid, 'PoTion: Pose MoTion Representation for Action Recognition', in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 1–10.

[77] J. Carreira and A. Zisserman, ‘Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset’, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 4724–4733.

[78] K. K. Reddy and M. Shah, ‘Recognizing 50 human action categories of web videos’, *Machine Vision and Applications*, vol. 24, no. 5, pp. 971–981, Jul. 2013.

[79] K. Soomro, A. Zamir and M. Shah, ‘UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild’, *CoRR*, Dec. 2012.

[80] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio and T. Serre, ‘HMDB: A large video database for human motion recognition’, in *International Conference on Computer Vision*, Nov. 2011, pp. 2556–2563.

[81] H. Jhuang, J. Gall, S. Zuffi, C. Schmid and M. J. Black, ‘Towards Understanding Action Recognition’, in *IEEE International Conference on Computer Vision (ICCV)*, Dec. 2013, pp. 3192–3199.

[82] M. Marszalek *et al.*, ‘Actions in Context’, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 2929–2936.

[83] M. D. Rodriguez, J. Ahmed and M. Shah, ‘Action MACH a spatio-temporal Maximum Average Correlation Height filter for action recognition’, in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, AK, USA: IEEE, Jun. 2008, pp. 1–8.

[84] K. Soomro and A. R. Zamir, ‘Action Recognition in Realistic Sports Videos’, in *Computer Vision in Sports*, T. B. Moeslund, G. Thomas and A. Hilton, Eds., Cham: Springer International Publishing, 2014, pp. 181–208.

[85] M. Rohrbach, S. Amin, M. Andriluka and B. Schiele, ‘A database for fine grained activity detection of cooking activities’, in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 1194–1201.

[86] G. Varol *et al.*, ‘Learning from synthetic humans’, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[87] W. Chen *et al.*, ‘Synthesizing training images for boosting human 3d pose estimation’, in *Fourth International Conference on 3D Vision (3DV)*, IEEE, 2016, pp. 479–488.

[88] H. Hattori, V. Naresh Boddeti, K. M. Kitani and T. Kanade, ‘Learning scene-specific pedestrian detectors without real data’, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3819–3827.

[89] J. Shotton *et al.*, ‘Real-time Human Pose Recognition in Parts from Single Depth Images’, *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, Jan. 2013.

[90] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, ‘CARLA: An Open Urban Driving Simulator’, *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, Nov. 2017.

[91] M. Mueller, V. Casser, J. Lahoud, N. Smith and B. Ghanem, ‘Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications’, *Int. Journal of Computer Vision*, pp. 902–919, 2018.

[92] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen and R. Vasudevan, ‘Driving in the Matrix: Can virtual worlds replace human-generated annotations for real world tasks?’, in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 746–753.

[93] C. R. de Souza, A. Gaidon, Y. Cabon and A. M. López, ‘Procedural Generation of Videos to Train Deep Action Recognition Networks’, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2594–2604.

[94] M. Khodabandeh, H. R. V. Joze, I. Zharkov and V. Pradeep, ‘DIY Human Action Dataset Generation’, *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1448–1458, 2018.

[95] D. Vázquez, A. M. López, J. Marín, D. Ponsa and D. Gerónimo, ‘Virtual and Real World Adaptation for Pedestrian Detection’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 797–809, Apr. 2014.

[96] B. Sun and K. Saenko, ‘From Virtual to Reality: Fast Adaptation of Virtual Object Detectors to Real Domains.’, in *British Machine Vision Conference (BMVC)*, 2014.

[97] E. Tzeng, J. Hoffman, K. Saenko and T. Darrell, ‘Adversarial Discriminative Domain Adaptation’, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 2962–2971.

[98] Y. LeCun, Y. Bengio and G. Hinton, ‘Deep learning’, *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[99] K. He, X. Zhang, S. Ren and J. Sun, ‘Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification’, in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV ’15, USA: IEEE Computer Society, Dec. 2015, pp. 1026–1034.

[100] K. Jarrett, K. Kavukcuoglu, M. Ranzato and Y. LeCun, ‘What is the best multi-stage architecture for object recognition?’, in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 2146–2153.

[101] V. Nair and G. E. Hinton, 'Rectified linear units improve restricted boltzmann machines', in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, Haifa, Israel: Omnipress, Jun. 2010, pp. 807–814.

[102] X. Glorot, A. Bordes and Y. Bengio, 'Deep Sparse Rectifier Neural Networks', in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Jun. 2011, ch. Machine Learning, pp. 315–323.

[103] D. E. Rumelhart, 'Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1', in J. L. McClelland and P. R. Group, Eds., Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[104] X. Glorot and Y. Bengio, 'Understanding the difficulty of training deep feedforward neural networks', in *Proceedings of the 13th Int. Conf. on Artificial Intelligence and Statistics*, Mar. 2010, pp. 249–256.

[105] C. M. Bishop, *Pattern Recognition and Machine Learning*, Second. New York: Springer, 2007.

[106] M. A. Cauchy, 'Méthode générale pour la résolution des systèmes d'équations simultanées', *Comptes Rendus Hebd. Séances Acad. Sci.*, vol. 25, pp. 536–538, 1847.

[107] D. P. Bertsekas, *Nonlinear Programming*, Second. Belmont, Mass: Athena Scientific, Sep. 1999.

[108] J. Kiefer and J. Wolfowitz, 'Stochastic Estimation of the Maximum of a Regression Function', *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, Sep. 1952.

[109] L. Bottou, 'Large-Scale Machine Learning with Stochastic Gradient Descent', in *Proceedings of COMPSTAT'2010*, Y. Lechevallier and G. Saporta, Eds., Physica-Verlag HD, 2010, pp. 177–186.

[110] S. Ioffe and C. Szegedy, 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', in *Proceedings of the 32nd Int. Conf. on Machine Learning*, ser. ICML'15, vol. 37, JMLR.org, 2015, pp. 448–456.

[111] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[112] M. Sokolova and G. Lapalme, 'A systematic analysis of performance measures for classification tasks', *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, Jul. 2009.

[113] C. Manning, P. Raghavan and H. Schuetze, ‘Introduction to Information Retrieval’, p. 581, 2009.

[114] Y. Yang and D. Ramanan, ‘Articulated Human Detection with Flexible Mixtures of Parts’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2878–2890, Dec. 2013.

[115] A. Toshev and C. Szegedy, ‘DeepPose: Human Pose Estimation via Deep Neural Networks’, in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA: IEEE, Jun. 2014, pp. 1653–1660.

[116] F. Wang, Y. Zhuang, H. Gu and H. Hu, ‘Automatic Generation of Synthetic LiDAR Point Clouds for 3-D Data Analysis’, *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 7, pp. 2671–2673, Jul. 2019.

[117] A. Feng, D. Casas and A. Shapiro, ‘Avatar Reshaping and Automatic Rigging Using a Deformable Model’, in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, ser. MIG ’15, New York, NY, USA: ACM, 2015, pp. 57–64.

[118] J. K. Hodgins, ‘Animating Human Motion’, *Scientific American*, vol. 278, no. 3, pp. 64–69, 1998.

[119] M. Gleicher, ‘Animation from observation: Motion capture and motion editing’, *ACM SIGGRAPH Computer Graphics*, vol. 33, no. 4, pp. 51–54, Nov. 1999.

[120] F. Thomas and O. Johnston, *The Illusion of Life: Disney Animation*, Revised, Subsequent. New York: Disney Editions, Oct. 1995.

[121] D. Jackèl, S. Neunreither and F. Wagner, *Methoden der Computeranimation* (eXamen.press). Berlin Heidelberg: Springer-Verlag, 2006.

[122] D. Holden, J. Saito and T. Komura, ‘A deep learning framework for character motion synthesis and editing’, *ACM Transactions on Graphics*, vol. 35, no. 4, 138:1–138:11, Jul. 2016.

[123] D. Holden, T. Komura and J. Saito, ‘Phase-functioned neural networks for character control’, *ACM Transactions on Graphics*, vol. 36, no. 4, 42:1–42:13, Jul. 2017.

[124] H. Zhang, S. Starke, T. Komura and J. Saito, ‘Mode-adaptive neural networks for quadruped motion control’, *ACM Transactions on Graphics*, vol. 37, pp. 1–11, Jul. 2018.

[125] J. K. Hodgins, W. L. Wooten, D. C. Brogan and J. F. O’Brien, ‘Animating human athletics’, in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’95, New York, NY, USA: Association for Computing Machinery, Sep. 1995, pp. 71–78.

[126] J. de Vries, *Learn OpenGL - Graphics Programming Learn Modern OpenGL Graphics Programming in a Step-by-Step Fashion.* 2020.

[127] M. Cordts *et al.*, 'The Cityscapes Dataset for Semantic Urban Scene Understanding', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Apr. 2016.

[128] S. Völker and T. Kistemann, '"I'm always entirely happy when I'm here!" Urban blue enhancing human health and well-being in Cologne and Düsseldorf, Germany', *Social Science & Medicine*, vol. 78, pp. 113–124, Feb. 2013.

[129] J. L. Kidder, 'Parkour: Adventure, Risk, and Safety in the Urban Environment', *Qualitative Sociology*, vol. 36, pp. 231–250, 2013.

[130] C. M. University, *Carnegie Mellon University - CMU Graphics Lab - motion capture library*, http://mocap.cs.cmu.edu/, Database, 2003.

[131] J. Redmon and A. Farhadi, 'YOLOv3: An Incremental Improvement', *arXiv:1804.02767*, Apr. 2018.

[132] J.-y. Bouguet, 'Pyramidal implementation of the Lucas Kanade feature tracker', *Intel Corporation, Microprocessor Research Labs*, 2000.

[133] N. Ma, X. Zhang, H.-T. Zheng and J. Sun, 'ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design', in *European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131.

[134] A. Y. Ng, 'Feature selection, L1 vs. L2 regularization, and rotational invariance', in *21st Int. Conf. on Machine Learning - ICML '04*, Banff, Alberta, Canada: ACM Press, 2004, p. 78.

[135] International Organization for Standardization, 'ISO 80000-2:2019 - Quantities and units — Part 2: Mathematics', Tech. Rep., Aug. 2019.

[136] S. Honari, P. Molchanov, S. Tyree, P. Vincent, C. Pal and J. Kautz, 'Improving Landmark Localization with Semi-Supervised Learning', in *Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, Oct. 2018.

[137] D. C. Luvizon, H. Tabia and D. Picard, 'Human Pose Regression by Combining Indirect Part Detection and Contextual Information', *Computers and Graphics*, vol. 85, pp. 15–22, Oct. 2017.

[138] A. Kendall, Y. Gal and R. Cipolla, 'Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics', in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA: IEEE, Jun. 2018, pp. 7482–7491.

[139] I. Loshchilov and F. Hutter, 'Decoupled Weight Decay Regularization', in *7th International Conference on Learning Representations (ICLR)*, Jan. 2019.

[140] D. P. Kingma and J. Ba, ‘Adam: A Method for Stochastic Optimization’, *arXiv:1412.6980 [cs]*, 2015.

[141] L. N. Smith, ‘Cyclical Learning Rates for Training Neural Networks’, in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Santa Rosa, CA, USA: IEEE, Mar. 2017, pp. 464–472.

[142] L. N. Smith, ‘A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay’, *arXiv:1803.09820 [cs, stat]*, Apr. 2018.

[143] K. Sun, B. Xiao, D. Liu and J. Wang, ‘Deep High-Resolution Representation Learning for Human Pose Estimation’, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 5686–5696.

[144] W. Yang, W. Ouyang, X. Wang, J. Ren, H. Li and X. Wang, ‘3D Human Pose Estimation in the Wild by Adversarial Learning’, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA: IEEE, Jun. 2018, pp. 5255–5264.

[145] W. Shan, H. Lu, S. Wang, X. Zhang and W. Gao, ‘Improving Robustness and Accuracy via Relative Information Encoding in 3D Human Pose Estimation’, in *Proceedings of the 29th ACM International Conference on Multimedia*, Virtual Event China: ACM, Oct. 2021, pp. 3446–3454.

[146] K. Gong, J. Zhang and J. Feng, ‘PoseAug: A Differentiable Pose Augmentation Framework for 3D Human Pose Estimation’, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Virtual, 2021, p. 10.

[147] K. Hara, R. Vemulapalli and R. Chellappa, ‘Designing Deep Convolutional Neural Networks for Continuous Object Orientation Estimation’, *arXiv:1702.01499 [cs]*, Feb. 2017.

[148] D. Yu, H. Xiong, Q. Xu, J. Wang and K. Li, ‘Continuous Pedestrian Orientation Estimation using Human Keypoints’, in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.