Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Vergleich und Implementierung mehrerer Handerkennungsalgorithmen mit maschinellem Lernen auf Basis von 2D- und 3D-Bilddaten

Martin Root

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Marc Toussaint
Betreuer/in:	Christian Jauch, M.Sc.
Beginn am:	2. April 2018
Beendet am:	2. Oktober 2018

Kurzfassung

Handerkennungsalgorithmen mit 2D/3D-Sensorik sind mit der Einführung des Kinect-Sensors im Jahr 2010 in vielen Forschungsarbeiten entwickelt und untersucht worden. Inzwischen gibt es eine Vielzahl verschiedener Ansätze und Methoden für die Handerkennung. Sie können die 3D-Gelenkpunktkoordinaten in der Hand ableiten und erlauben somit Rückschlüsse über die ausgeführte Handpose zu ziehen. Die besten Ergebnisse erzielen die Verfahren mit dem Einsatz von maschinellem Lernen. Hierzu werden die Handerkennungsalgorithmen mithilfe eines Handposendatensatzes trainiert. Diese Arbeit gibt einen Überblick über den aktuellen Stand der Handerkennungsalgorithmen. Es werden verschiedene Verfahren miteinander verglichen und gegenübergestellt. Neben einer selbst entwickelten Methode, werden drei weitere Handerkennungsalgorithmen implementiert und auf einem Handposendatensatz und einem Testszenario in der manuellen Montage ausgewertet. Mithilfe des Testszenarios wurde bewertet, ob die Handerkennungsalgorithmen auch in dem Anwendungsgebiet der manuellen Montage zuverlässig eingesetzt werden können. Die Evaluation hat aufgezeigt, dass die Handerkennungsalgorithmen auf dem Testdatensatz durchaus genaue Resultate erzielen konnten. Mit der selbst entwickelten Methode ist im Testdatensatz auf der x- und y-Ebene die höchste Genauigkeit erreicht worden. Bei der Auswertung des Testszenarios in der manuellen Montage ergab sich, dass alle der verglichenen Handerkennungsalgorithmen zu ungenaue Resultate erzielten um sinnvolle Rückschlüsse auf die Handpose zu ziehen. Aktuell öffentlich verfügbare Handposendatensätze unterscheiden sich deutlich von den Handposen im Testszenario. Die Interaktion beider Hände mit Bauteilen und Werkzeug führt zu wesentlich komplexeren Posen. Diese Diskrepanz zwischen Trainingsdatensatz und Testszenario stellt die größte Problematik für die Handerkennungsalgorithmen dar.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ANN Artificial Neural Network 10

CNN Convolutional Neural Network 5

FC-Layer Fully-Connected-Layer 13

ICP Iterative Closest Point 5

MP-Layer Max-Pooling-Layer 33

PSO Particle Swarm Optimisation 27

R-CNN Region Based Convolutional Neural Network 24

ResNet Residual Network 5

1 Einleitung

Jeden Tag verwenden wir unsere Hände um mit der Umgebung zu interagieren. Dabei kann die Hand auf viele verschiedene Weisen genutzt werden, selbst Kommunikation durch Handgesten ist möglich. Aktuell existiert bereits eine Vielzahl an Methoden, die Hände zur Steuerung verwenden. Die genaue Erkennung der Handpose spielt dabei eine wichtige Rolle. In den letzten Jahren haben sich Handerkennungsalgorithmen deshalb rapide weiterentwickelt. Durch den Einsatz von maschinellem Lernen sind die Handerkennungsalgorithmen robuster und präziser geworden. Vor allem im Bereich Mensch-Computer-Interaktion haben sich neue vielversprechende Anwendungsgebiete, wie beispielsweise Robotik oder Virtual Reality, aufgetan [MZ15].

Ziel dieser Handerkennungsalgorithmen ist es, die Gelenkpunktkoordinaten der Hand abzuleiten, um Rückschlüsse auf die Handpose ziehen zu können. Inzwischen gibt es eine Vielzahl an verschiedenen Handerkennungsalgorithmen, die auf maschinellem Lernen basieren und mithilfe von 2D/3D-Bilddaten die Handgelenkpunktkoordinaten bestimmen können. Die 3D-Sensorik hat sich ebenfalls weiterentwickelt. Seit der Einführung des ersten Kinect-Sensors von Microsoft im Jahr 2010, verwenden die meisten Handerkennungsalgorithmen 3D-Sensorik und konnten auf diese Weise die Genauigkeit der Methoden erhöhen. Hierfür werden mithilfe von 3D-Sensoren Tiefenbilder aufgenommen. Für jeden Pixel wird dabei zusätzlich ein Tiefenwert, der angibt in welcher Entfernung sich dieser von dem Sensor befindet, aufgenommen. Die erhaltenen 3D-Gelenkpunktkoordinaten erlauben es Handposen erkennen zu können und so Rückschlüsse auf die ausgeführte Geste zu ziehen. Die Handerkennungsalgorithmen teilen sich in zwei Funktionen auf: Handsegmentierung und die Ableitung der Gelenkpunkte. Ein Anwendungsgebiet für solche Handerkennungsalgorithmen ist die manuelle Montage. Mithilfe dieser kann beispielsweise ein Assistenzsystem entworfen werden, dass die komplexen Arbeitsschritte in der Montage verfolgen und den Arbeiter auf diese Weise unterstützen soll. Es können vergessen Montageschritte aufgezeigt und auf Fehler während der Montage hingewiesen werden.

Ziel dieser Arbeit ist es, einen Überblick über vorhandene Handerkennungsalgorithmen zu geben und ausgewählte Methoden auf ihre Robustheit und Genauigkeit zu untersuchen. Im Speziellen soll die Einsetzbarkeit solcher Verfahren im Szenario eines Assistenzsystems für die manuelle Montage untersucht werden. Dabei wird neben den verglichenen Handerkennungsalgorithmen ein selbst entwickeltes Verfahren implementiert und vorgestellt. Der Beitrag dieser Arbeit setzt sich wie folgt zusammen:

1. Überblick und Vergleich aktueller Handerkennungsalgorithmen
2. Auswahl und Implementierung von mehreren Algorithmen
3. Konzeption und Entwicklung eines eigenen Handerkennungsalgorithmus
4. Evaluation der Algorithmen auf einem Handposendatensatz
5. Konzeption und Evaluation eines Testszenarios im Bereich der manuellen Montage

2 Grundlagen

In diesem Kapitel werden die Grundlagen für die in der Arbeit verwendeten neuronalen Netzwerkarchitekturen und Datenformate vorgestellt. Dabei werden sowohl mathematische Grundlagen für das Trainieren eines Netzwerks als auch Eigenschaften der Netzwerkarchitekturen beschrieben.

2.1 Artificial Neural Network

Artificial Neural Network (ANN) sind nach dem Vorbild des menschlichen Gehirns entworfen worden. Wie im Gehirn werden Informationen durch Neuronenverbände geleitet und ausgewertet. Die Netzwerkarchitektur wird durch ein Input-Layer, ein oder mehrere Hidden-Layer und ein Output-Layer definiert. Zuerst werden die Eingabedaten an das Input-Layer gegeben, in den Hidden-Layer verarbeitet und im Output-Layer ausgegeben. In Abbildung 2.1 ist ein solcher Aufbau dargestellt.

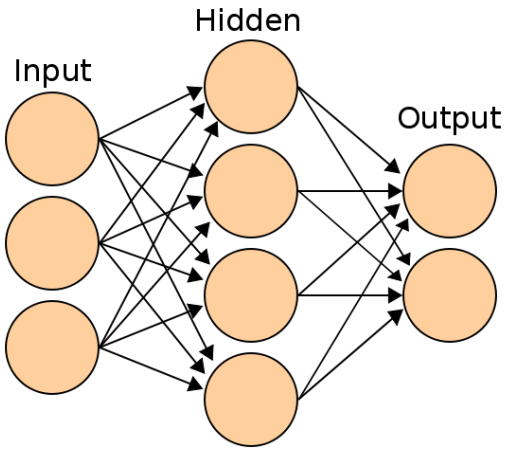


Abbildung 2.1: Aufbau eines Artificial Neural Networks. Ein ANN besteht aus einem Input-Layer gefolgt von einem oder mehreren Hidden-Layer. Den Abschluss bildet ein sogenanntes Output-Layer. Die Kreise symbolisieren die Neuronen und die Pfeile ihre Verbindungen miteinander [wik].

Ein Layer besteht aus Neuronen, welche mit dem nächsten Layer verbunden sind. Für jede dieser Verbindungen existiert eine Gewichtung, die angibt, wie stark diese Verbindung zwischen den Neuronen gewertet werden soll. Ein Neuron kann dabei als Verarbeitungseinheit betrachtet werden, welche eine Aktivierungsfunktion beinhaltet, die basierend auf dem Eingangswert entscheidet, ob das Neuron aktiviert wird. Damit das ANN für den gegebenen Input das gewünschte Ergebnis ausgibt, muss eine passende Gewichtung für die Neuronenverbindungen gefunden werden. Dieser Vorgang

wird als "Trainieren des Netzwerkes" bezeichnet. Hierfür wird ein sogenannter Lernalgorithmus definiert. Ein Beispiel ist der Back-Propagation Algorithmus. Durch ihn, kann das Netzwerk aus den Fehlern während der Trainingsphase lernen. Hierfür werden gelabelte Daten benötigt, das heißt für jedes Bild im Datensatz, ist der reale Wahrheitswert, der bestimmt werden soll, angegeben. Die Funktion für das Aktualisieren der Parameter ist wie folgt definiert [KRSB18]:

$$\Theta_{ij}^{t+1} = \Theta_{ij}^{t} + \eta \frac{\partial E}{\partial \Theta_{ij}}. \tag{2.1}$$

.

Die Gewichte des Netzwerks (Θ) werden basierend auf der Ausgabe des vorherigen Trainingsdurchgangs (^{t}) und des Gradienten der Fehlerfunktion ($\frac{\partial E}{\partial \Theta_{ij}}$) unter Rücksicht der Gewichte des Netzwerks (Θ) aktualisiert. Die Fehlerfunktion beschreibt dabei, wie stark die Ausgabe des Netzwerks von dem richtigen Ergebnis abweicht. Der Gradient gibt die Steigung innerhalb dieser Funktion an, er zeigt in die Richtung des stärksten Anstiegs. Auf diese Weise kann bei jedem Update der Parameter ein Schritt in die richtige Richtung (basierend auf der Fehlerfunktion) vorgenommen werden. Der Parameter η gibt an, wie groß dieser Schritt sein soll. Dabei können viele unterschiedliche Fehlerfunktionen eingesetzt werden. Ein Beispiel für eine solche Funktion ist die "kleinster mittlerer quadratischer Fehler Funktion" (Least Mean Square Error). Sie ist wie folgt definiert [KRSB18]:

$$\begin{aligned} E &= \frac{1}{n} \sum_{n} (y_n - p_n)^2, \\ p_i &= \sum_{j} \Theta_{ij} x_j. \end{aligned} \tag{2.2}$$

Der Parameter p steht für den Ausgabewert des Netzwerks basierend auf den Gewichten Θ und Eingabewerten x. Parameter y beschreibt den korrekten Ausgabewert und Parameter n steht für die Anzahl der Neuronen im Output-Layer, die beispielsweise stellvertretend für die Anzahl der zu erkennenden Klassen stehen. Auf diese Weise gibt die Fehlerfunktion den mittleren quadratischen Fehler zurück. Mit den gegebenen Funktionen aus 2.1, 2.2 kann der Gradient der Fehlerfunktion für das Output-Layer L und die Hidden-Layer berechnet und daraufhin alle Parameter rückwirkend aktualisiert werden. Dieser Vorgang wird mehrere Male wiederholt bis die Parameter des Netzwerks sich nicht mehr signifikant verändern (der Gradient wird verschwindend gering) [KRSB18]. Sind ausreichend Daten vorhanden gewesen und eine passende Netzwerkarchitektur gewählt, ist das Netzwerk trainiert und für die Vorhersage optimiert. Das heißt, es kann auch für neue Eingangsdaten die richtigen Ausgabewerte liefern.

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) zeichnen sich durch mehrfach wiederholende Blöcke an Neuronen aus, die lokalisierte Eigenschaften anhand der Eingabe ableiten können. Sie werden vor allem bei mehrdimensionalen Daten angewendet wie Bilder oder Videos. Ein Hauptunterschied zu einem standardmäßigen neuronalen Netz ist, dass die Layer im CNN wie ein mehrdimensionaler Filter aufgebaut sind. Diese Filter werden mit dem Input des Layers gefaltet und auf diese Weise können Merkmale extrahiert werden. Ein CNN besteht aus mehreren Layern, die miteinander

verbunden sind. Der Aufbau des Netzwerks ist ähnlich zum ANN, es gibt auch hier das Input-Layer, die Hidden-Layer und das Output-Layer. In Abbildung 2.4 ist ein CNN dargestellt. Das Hauptmerkmal eines CNN sind die Convolutional-Layer, es besitzt aber weitere Layer welche folgend näher erläutert werden [KRSB18].

2.2.1 Convolutional-Layer

Das Convolutional-Layer ist die wichtigste Komponente innerhalb eines CNN. Es besteht aus Filtern (auch Kernel genannt), welche mit einem Input gefaltet werden um sogenannte Activation-Maps zu erzeugen. In Abbildung 2.2 ist ein solcher Filter dargestellt. Ein Filter ist dabei eine Zelle mit mehreren diskreten Zahlen. In Abbildung 2.2 ist ein 3x3 Filter verwendet worden, dieser wird über das Eingabebild gelegt und die entsprechen Position mit dem Filter elementweise multipliziert. Der Wert der Activation-Map wird durch Addition mit den Multiplikationen errechnet. Zusätzlich kann festgelegt werden, in welchem Abstand der Filter über das Eingabebild geschoben wird. Hierfür wird ein sogenannter Stride definiert. Wenn der Filter über den Rand des Eingabebilds kommt, können Funktionen definiert werden, welche den Rand des Eingabebilds erweitern, sodass der Filter hineinpasst, oder es wird festgelegt, dass der Filter den Rand des Eingabebilds nicht überschreiten darf. Durch die Verwendung eines solchen Convolutional-Layers können einzelne Eigenschaften im Bild gelernt werden. Üblicherweise werden mehrere Convolutional-Layer hintereinander verwendet. Während das erste Layer noch als Input das Eingabebild verwendet, dienen den anderen Convolutional-Layern als Input jeweils die erzeugten Activation-Maps des vorherigen Layers. In den ersten Layern werden Low-Level Features wie Kreise, Kurven, Linien oder Kanten erkannt und in den tieferen Layer können High-Level Features, wie die Form der Augen oder die Form des Kopfes, erkannt werden. Dabei setzen sich diese High-Level Features aus den zuvor gelernten Low-Level Features zusammen. Die Convolutional-Layer in einem CNN sind dafür zuständig, die Eigenschaften im Bild zu extrahieren und ermöglichen somit auf Basis dieser Eigenschaften korrekte Vorhersagen treffen zu können [KRSB18].

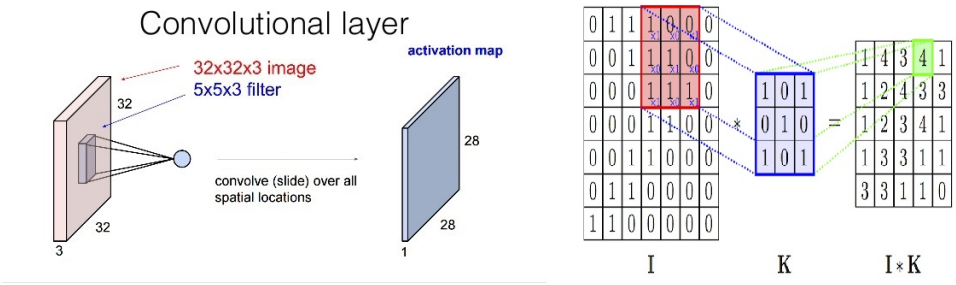


Abbildung 2.2: Aufbau eines Convolutional-Layers. Ein Filter (K) wird über den Input(I) geschoben. Dadurch wird eine Activation-Map erstellt. Aus einem 32x32x3 Bild und einem 5x5x5 Filter wird eine 28x28x1 große Activation-Map erzeugt. Auf der rechten Seite, ist dargestellt, wie eine Position innerhalb der Activation-Map berechnet wird [Fol16; Spa].

2.2.2 Pooling-Layer

Das Pooling-Layer ist ähnlich aufgebaut wie ein Convolutional-Layer. Auch hier wird ein Filter definiert, welcher im Gegensatz zu dem Convolutional-Filter keine Zahlenwerte beinhaltet. Oft verwendete Pooling Funktionen sind Max-Pooling und Average-Pooling. Beim Max-Pooling wird der Maximalwert im Filterbereich, beim Average-Pooling der Durchschnittswert zurückgegeben. In Abbildung 2.3 ist eine Max-Pooling Operation mit der Filtergröße 2x2 und einem Stride von 2 dargestellt. Das Pooling-Layer dient dazu, die Größe des Eingabebilds zu reduzieren und verringert somit die Anzahl der Parameter, die Dimension der Eingabe und die Anzahl der benötigten Berechnungen. Meist wird ein Pooling-Layer zwischen Convolutional-Layern eingesetzt [KRSB18].

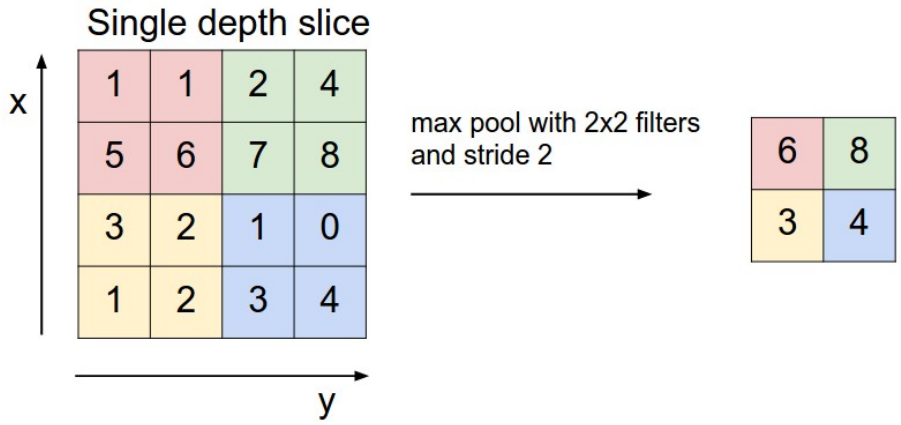


Abbildung 2.3: Aufbau eines Max-Pooling-Layer (MP-Layer). Hier wird ein Filter der Größe 2x2 über das Eingabebild gelegt und jeweils der größte Wert im Filterbereich in das Ausgabebild geschrieben [CS2].

2.2.3 Fully-Connected-Layer

Ein Fully-Connected-Layer (FC-Layer) ist genau genommen ein Convolutional-Layer mit einem 1x1 Filter. Das heißt jedes Neuron im vorherigen Layer ist mit jedem Neuron im FC-Layer verbunden. Meist befinden sich FC-Layer am Ende des Netzwerks und helfen bei der Klassifikation beziehungsweise Regression. Sie dienen dazu die gelernten Features zusammenzufassen und daraus das Ergebnis als Output abzuleiten. Es können nicht lineare Kombination an Features gelernt werden, welche jeweils zu einem bestimmten Ergebnis führen.

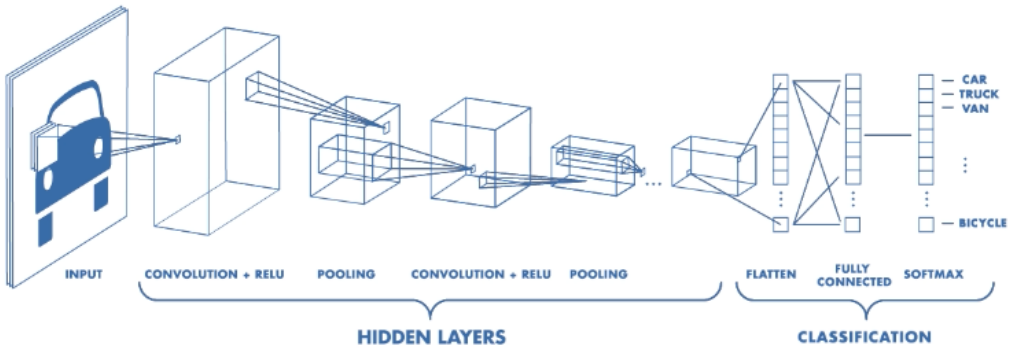


Abbildung 2.4: Aufbau eines Convolutional Neural Network. Das CNN besteht aus einem Input-Layer gefolgt von Hidden-Layer und den Abschluss bildet ein Output-Layer bestehend aus einem Flatten-Layer und FC-Layer für die Klassifikation [Mat].

2.3 Residual Network

Residual Networks (ResNets) basieren auf einer überarbeiteten CNN-Architektur. Sie sind mit dem Ziel entworfen worden, tiefe CNNs verwenden zu können, ohne dass sich die Performance sowohl beim Training als auch beim Testdatensatz verschlechtert. Dies ist der Fall, bei der Verwendung von tiefen CNNs mit herkömmlichen CNN-Architekturen, die nur mehrere Convolutional-Layer hintereinander stapeln. In Abbildung 2.5 ist dieses Verhalten dargestellt.

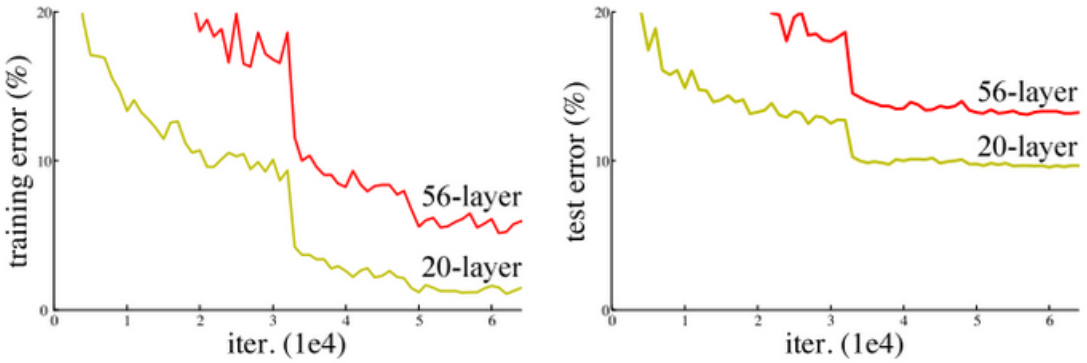


Abbildung 2.5: Erhöhung der Tiefe herkömmlicher CNNs führt zur schlechterer Performance. Sowohl der Trainings- als auch der Testfehler steigen [HZRS16].

Werden zu viele Layer verwendet, steigt sowohl der Trainings- als auch der Testfehler. Diese Eigenschaft ist darauf zurückzuführen, dass der Gradient bei solch tiefen Netzwerken verschwindend gering wird bevor die Merkmale gelernt werden konnten und somit das Training des CNN nicht mehr vorangehen kann [HZRS16; KRSB18]. Um dies zu umgehen, ist die ResNet-Architektur entworfen

worden. Sie besteht aus mehreren sogenannten Residual Blocks. Hierzu werden zu den Convolutional-Layer zusätzlich Skip Connections hinzugefügt. In Abbildung 2.6 ist dies bildlich dargestellt. Diese Skip Connections stellen eine sogenannte Identity Connection (Identitätsverbindung) dar, welche die Convolutional-Layer umgeht. Ein solcher Residual Block ist in Abbildung 2.6 dargestellt.

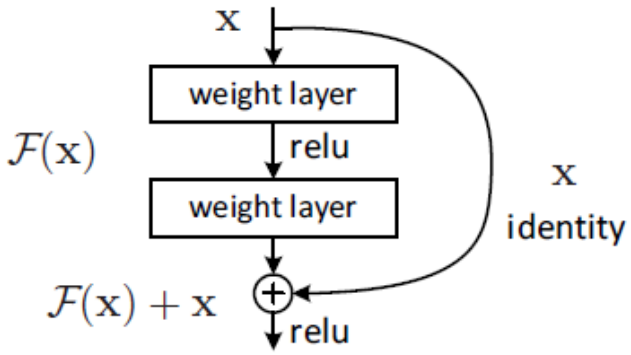


Abbildung 2.6: Residual Block eines ResNet. Zu jedem Convolutional-Layer wird eine sogenannte Skip Connection hinzugefügt. [HZRS16].

Die Transformation setzt sich aus den Transformationen der Layer $F(x)$ und der Identitätsfunktion x (repräsentiert den Eingangswert in den Residual Block) zusammen. In der Praxis, erlaubt eine solche Architektur wesentlich tiefere Netzwerke zu trainieren [KRSB18]. In Abbildung 2.7 ist der Vergleich zu einer herkömmlichen CNN-Architektur graphisch dargestellt.

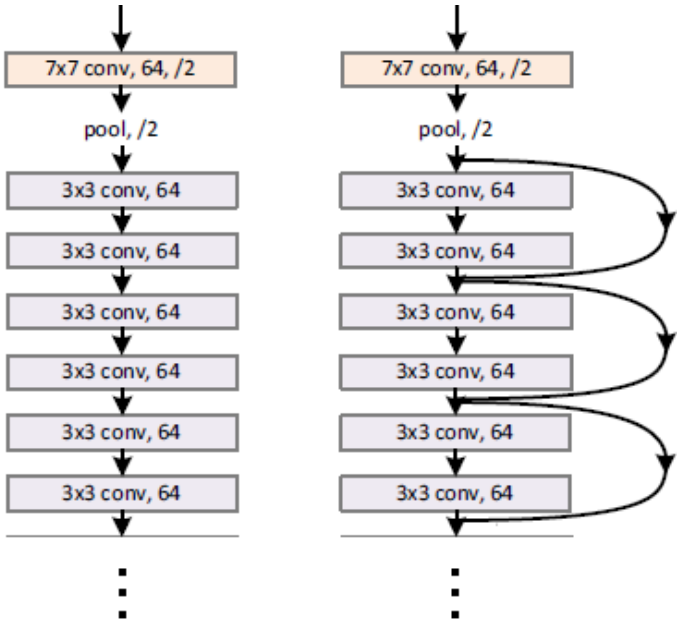


Abbildung 2.7: Architektur eines ResNets im Vergleich zu herkömmlichen CNN-Architekturen. Auf der linken Seite ist ein herkömmliches CNN abgebildet, auf der rechten Seite ein ResNet mit Skip Connections [HZRS16].

2.4 Verwendete Datenformate

2.4.1 Tiefenbild

Ein Tiefenbild stellt die Tiefeninformationen einer Szene in einem zweidimensionalem Bild dar. Dabei beschreibt jeder einzelne Pixelwert die Tiefe, das heißt wie weit weg dieser Pixel sich von der Kamera befindet. Der Tiefenwert wird meist in Millimetern (Weltkoordinaten) angegeben. Um die 3D-Koordinaten aus einem Tiefenbild abzuleiten, sind die Sensorparameter notwendig. Auf folgende Weise können die Pixelkoordinaten in Weltkoordinaten umgewandelt werden.

$$\begin{aligned} worldX &= (pixelX - Bildbreite/2) * Tiefenwert/fx) \\ worldY &= (pixelY - Bildhoehe/2) * Tiefenwert/fy) \\ worldZ &= Tiefenwert \end{aligned} \tag{2.3}$$

Um Weltkoordinaten in Pixelkoordinaten umzuwandeln, wird folgende Funktion verwendet.

$$\begin{aligned} pixelX &= fx * worldX/Tiefenwert + Bildbreite/2 \\ pixelY &= fy * worldY/Tiefenwert + Bildhoehe/2 \end{aligned} \tag{2.4}$$

Die Parameter fx und fy stehen für die Brennweite der Linse im Sensor. In Abbildung 2.8 ist ein solches Tiefenbild dargestellt. Nahe Pixel werden heller und Pixel, die sich weiter weg befinden, dunkler dargestellt. In diesem Beispiel wurde eine Graustufendarstellung gewählt. Es können auch Farbkodierungen zur Visualisierung angewendet werden.

Abbildung 2.8: Tiefenbild eines Tiefensensors. Nahe Pixel sind heller dargestellt, während Pixel, die sich weiter weg befinden, dunkler dargestellt werden [pra].

2.4.2 Punktwolke

Aus einem Tiefenbild kann eine dreidimensionale Punktwolke berechnet werden. Dazu werden mithilfe der Sensorparameter die 3D-Koordinaten abgeleitet und graphisch abgebildet. In Abbildung 2.9 ist eine dreidimensionale Punktwolke dargestellt. Sie helfen die 3D-Eigenschaften einer Szene besser visualisieren zu können und werden deshalb oft bei Verwendung von Tiefenbildern als

Visualisierung eingesetzt. Ein Tiefenbild erlaubt keine komplette 3D-Repräsentation der Szene. Aus dem zweidimensionalen Tiefenbild können für die Pixel, die im Vordergrund liegen und nicht durch andere Objekte überdeckt werden, die Tiefeninformationen extrahiert werden. Somit lässt sich keine komplette 3D-Repräsentation der Szene erstellen. Es wird auch von einer 2.5D-Repräsentation gesprochen [LJC06].

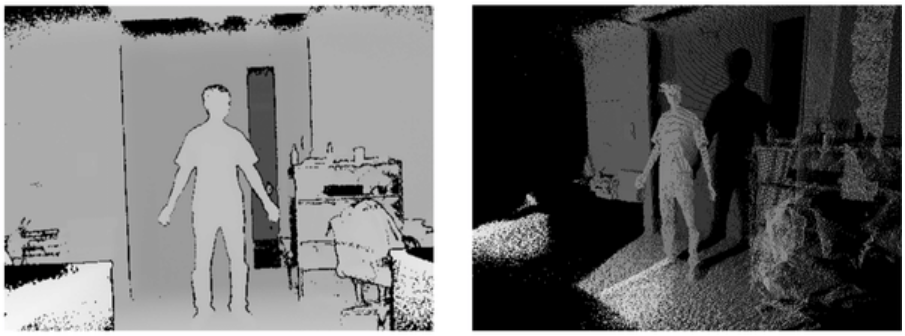


Abbildung 2.9: Tiefenbild und zugehörige Punktwolke. Aus dem Tiefenbild (links) ist die Punktwolke (rechts) erzeugt worden [Din15].

2.4.3 Heatmap

Eine Heatmap ist eine graphische Darstellungsform, in welcher Daten und deren Werte in einer mehrdimensionalen Definitionsmenge farblich dargestellt werden. Sie wird meist zu dem Zweck eingesetzt, große Datenmengen visuell schnell erfassbar zu machen [Onp]. In Abbildung 2.10 ist eine solche Heatmap dargestellt. In diesem Beispiel wird der kleine Finger gesucht, dabei werden die Pixel nach der Wahrscheinlichkeit, ob sich sich in dieser Region befinden, eingefärbt. Durch die Farbdarstellung lassen sich die Übergange und die Wahrscheinlichkeitswerte einfacher darstellen.

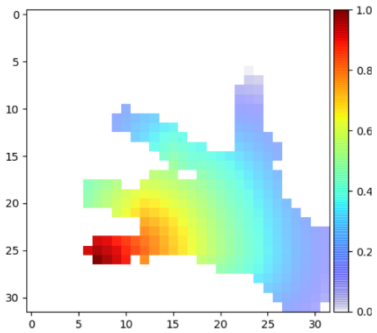


Abbildung 2.10: Darstellung einer Heatmap. Es wird der kleine Finger gesucht. Die Wahrscheinlichkeit, ob es sich bei einem Pixel hierbei um die kleine Finger Region handelt, wird durch eine Farbkodierung (rechts im Bild) ausgedrückt. Rot steht für sehr wahrscheinlich und blau für sehr unwahrscheinlich [WFOY18].

3 Stand der Technik

In diesem Kapitel wird der aktuelle Stand der Sensortechnik, Handsegmentierung, Handerkennungsalgorithmen und verfügbaren Datensätzen beschrieben. Dabei werden mehrere Verfahren miteinander verglichen und deren Ergebnisse auf etablierten Handposen-Datensätzen vorgestellt.

3.1 Übersicht

In dieser Arbeit werden Handerkennungsalgorithmen untersucht, die mithilfe von maschinellem Lernen die 3D-Gelenkpositionen der Hand im Eingabebild bestimmen können. Es werden Methoden in Betracht gezogen, die in Echtzeit operieren können. Grundsätzlich lässt sich zwischen markerbasierten- und markerlosen Verfahren unterscheiden. Markerbasierte Methoden verwenden zusätzliche Erkennungsmerkmale, um die Gelenkpositionen der Hand bestimmen zu können. Wang und Popović [WP09] verwenden ein mit verschiedenen Farben gefärbten Handschuh, um Rückschlüsse auf die Handpose ziehen zu können. In dieser Arbeit werden markerlose Methoden untersucht, da sie flexibler sind und keinerlei Einschränkungen für die Nutzer bedeuten. Der Ablauf solcher Handerkennungsalgorithmen lässt sich in drei Schritte untergliedern, wie in Abbildung 3.1 dargestellt. Im ersten Schritt wird die Hand im Bild lokalisiert und die Handregion zurückgegeben. Hierfür existieren verschiedene Herangehensweisen, welche in diesem Kapitel vorgestellt werden. Der zweite Schritt besteht aus der Extraktion der Merkmale aus der erhaltenen Handregion im Eingabebild. In diesem Punkt unterscheiden sich die meisten Handerkennungsalgorithmen voneinander. Im letzten Schritt werden anhand der extrahierten Merkmale die Gelenkpositionen der Hand abgeleitet. Das Ergebnis stellen die dreidimensionalen Gelenkpunktkoordinaten der Hand dar. In Abbildung 3.6 ist ein Beispiel solcher Gelenkpunkte dargestellt.

Die markerlosen Handerkennungsalgorithmen teilen sich in drei verschiedene Gruppen auf:

- **Datenbasierte Methoden:** Diese Methoden lernen auf Basis einer großen Datenmenge direkt anhand eines Eingabebilds die Handpose vorherzusagen.
- **Modellbasierte Methoden:** Diese Methoden verwenden ein Handmodell und versuchen anhand des Eingabebilds die dazu am besten passende Handmodellpose zu finden.
- **Hybride Methoden:** Verbinden modellbasierte und datenbasierte Methoden miteinander.

Der Fokus wird auf die datenbasierten Verfahren gelegt, da diese aktuell die besten Ergebnisse erzielen [YYGK17] und die wenigsten Einschränkungen an die aufzunehmende Szene und Testperson legen. Die meisten Handerkennungsalgorithmen verwenden 3D-Sensorik für die Aufnahme der Eingabebilder, durch die zusätzlichen Tiefeninformationen lassen sich genauere Rückschlüsse auf die 3D-Gelenkposition ziehen. Verfahren, die nur auf RGB-Bildinformationen zurückgreifen, haben weniger Informationen und Daten zur Verfügung und werden deshalb nicht betrachtet.

Die anschließenden Abschnitte geben einen Überblick über den Stand der Technik zu dem eben spezifizierten Bereich der Handerkennungsalgorithmen, sowie der verwendeten 3D-Sensorik, der Handsegmentierung und Datensätzen für die Handerkennung.

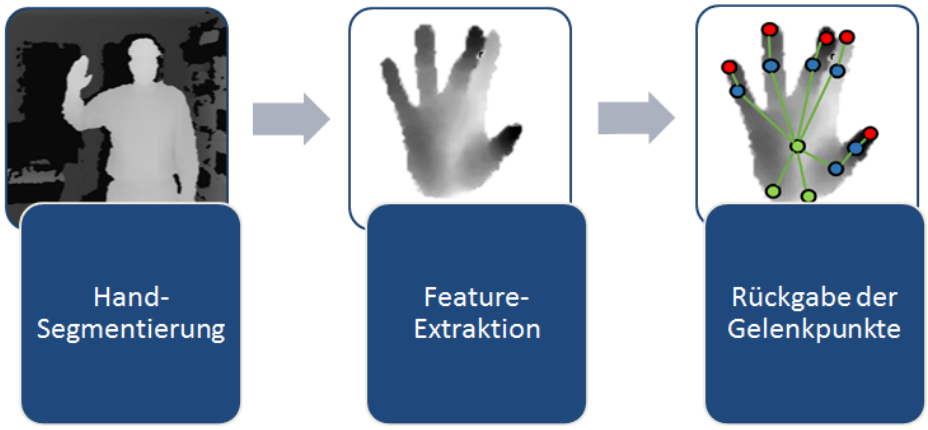


Abbildung 3.1: Generelle Funktionsweise der Handerkennungsalgorithmen. In dem ersten Schritt erfolgt die Segmentierung der Hand, im zweiten Schritt die Extraktion der Merkmale. Am Ende werden anhand der extrahierten Merkmale die 3D-Positionen der Gelenkpunkte abgeleitet. [NWNT17; pra].

3.2 3D-Sensorik

Mit der Einführung des Kinect-Sensors von Microsoft im Jahr 2010 ist der erste preiswerte RGB-D Sensor vorgestellt worden. Seitdem wird in der Forschung mehr auf 3D-Sensoren zurückgegriffen [Zha12]. Dies gilt auch für den Bereich der Handerkennungsalgorithmen, welcher inzwischen zum Großteil 3D-Sensorik einsetzt. Ein Überblick über Handerkennungsverfahren, welche den Kinect Sensor verwenden, ist in [CYL16] zu finden. Die hier genutzten Anwendungsgebiete reichen von Roboternavigation über Handgestenerkennung, Hand-Objekt Interaktion und Handtracking.

3D-Sensoren liefern neben dem normalen RGB-Bild ebenfalls Tiefenwerte zurück. Durch diese zusätzlichen Informationen lassen sich zuverlässigere und genauere Rückschlüsse auf die Gelenkpositionen in der Hand schließen. Genaueres dazu ist im Abschnitt ”Handsegmentierung” erläutert. Der erste Kinect Sensor selbst basiert auf dem Structured Light Prinzip. Inzwischen gibt es weitere Technologien, die eingesetzt werden. Im Folgenden werden diese näher erläutert.

3.2.1 Structured Light Prinzip

Beim Structured Light Prinzip wird eine Sequenz an bekannten Lichtmustern ausgesendet. Treffen diese auf ein Objekt, werden sie je nach geometrischer Form des Objekts deformiert. Dieses Objekt wird zusätzlich aus einer anderen Richtung von einer Kamera aufgenommen. Mithilfe des deformierten- und des originalen Musters können die Tiefenwerte berechnet werden [SLK15]. In Abbildung 3.2 ist der Aufbau eines 3D-Sensors, welcher das Structuerd Light Prinzip verwendet, dargestellt. Es wird ein Projektor benötigt, der das Lichtmuster aussendet und eine Kamera, die

das deformierte Lichtmuster aufnimmt. Das Structured Light Prinzip kam im ersten Kinect Sensor zum Einsatz und ist somit das erste Verfahren, welches in einem preiswerten 3D-Sensor verwendet worden ist. Ein weiterer 3D-Sensor, der auf dem Structured Light Prinzip basiert, ist beispielsweise der Asus XtionPro (siehe Tabelle 3.1).

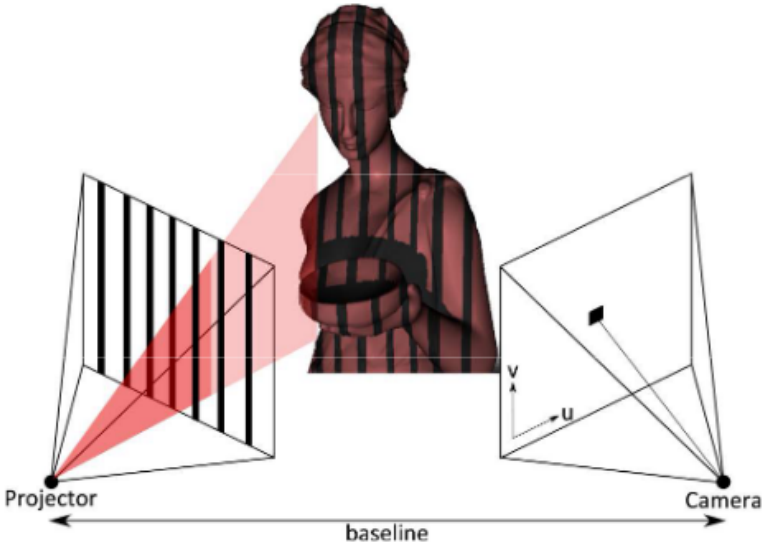


Abbildung 3.2: Darstellung des Structured Light Prinzips. Es wird ein Projektor und eine Kamera verwendet. Der Projektor sendet ein Lichtmuster aus. Die Kamera nimmt die Veränderung des Lichtmusters auf, wenn dieses auf ein Objekt trifft. [SLK15].

3.2.2 Time of Flight Prinzip

Das Time of Flight Prinzip basiert auf einem Laser, der einen Impuls an Licht aussendet. Dieser wird von einem Objekt reflektiert und zum Sensor zurückgeworfen. Sobald der Laserstrahl auf den Detektor im Sensor trifft, wird die verstrichene Zeit gemessen. Auf diese Weise kann mithilfe der gegebenen Lichtgeschwindigkeit die Entfernung, die der Laserstrahl zurückgelegt hat, berechnet werden [SLK15]. In Abbildung 3.3 ist dieser Ablauf dargestellt. Ein oft verwendeter 3D-Sensor, der auf dem Time of Flight Prinzip basiert, ist der Kinect V2 Sensor (siehe Tabelle 3.1). Die meisten neueren Sensoren verwenden inzwischen das Time of Flight Prinzip oder Active Stereo, da hierdurch eine höhere Genauigkeit und Reichweite bei der Messung erreicht werden kann [SH].

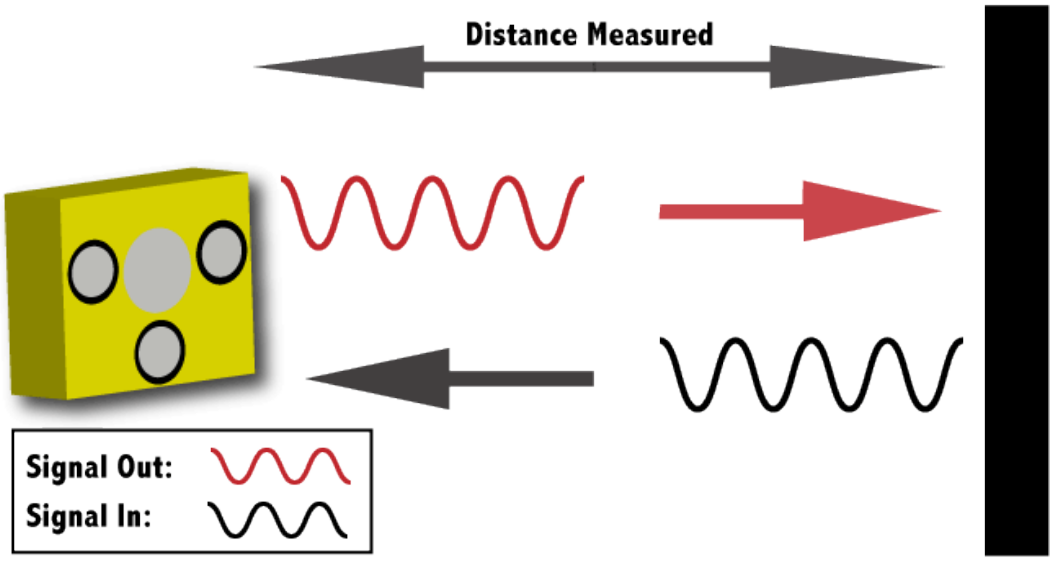


Abbildung 3.3: Darstellung des Time of Flight Prinzips. Ein Laser sendet einen Lichtstrahl aus, welcher reflektiert und von einem Detektor im Sensor erkannt wird. Dabei wird die vergangene Zeit gestoppt und mithilfe der Lichtgeschwindigkeit die zurückgelegte Entfernung berechnet [Ter].

3.2.3 Active Stereo Vision Prinzip

Das Active Stereo Vision Prinzip baut auf dem Stereo Vision Prinzip auf. Zwei Kameras nehmen ein Bild auf und durch Abgleich der zwei Bilder kann die 3D-Position mithilfe von Triangulation berechnet werden. Dabei können Probleme auftreten, wenn das Objekt keinerlei Textur besitzt, deshalb sendet das Active Stereo Vision Prinzip, gegenüber Stereo Vision, zusätzlich ein Lichtmuster aus. Dadurch kann Textur auf texturlose Objekte gebracht und die 3D-Position genauer berechnet werden [FVR+17]. In Abbildung 3.4 ist der Aufbau eines solchen Sensors dargestellt. Ein Beispiel für einen Sensor, der auf dem Active Stereo Vision Prinzip basiert, ist der Intel Realsense D435 (siehe Tabelle 3.1). Das Active Stereo Prinzip erlaubt eine deutlich höhere Genauigkeit und Reichweite als bei Time of Flight und Structured Light, allerdings ist die dahinter liegende Berechnung deutlich komplexer und rechenintensiver [FVR+17]. Deshalb sind erst seit kurzem preisgünstige Sensoren mit dieser Technologie für Privatkunden erhältlich und werden inzwischen immer öfters in der Forschung eingesetzt.

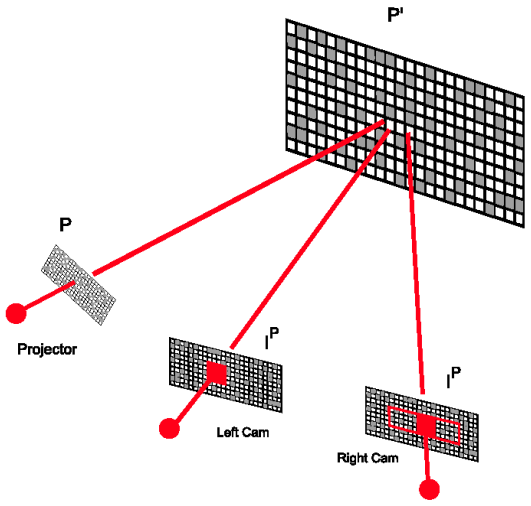


Abbildung 3.4: Darstellung des Active Stereo Vision Prinzips. Ein Lichtmuster wird durch Projektor P auf ein Objekt gesendet. Dies wird von zwei Kameras aufgenommen. Für das Stereo Matching wird der kleine rote Block im linken Bild gegen eine Reihe von Blöcken mit dem selben vertikalen Abstand aus dem rechten Bild verglichen und die 3D-Position berechnet [Kon10].

3.2.4 Vergleich verschiedener 3D-Sensoren

In Tabelle 3.1 ist Überblick über 3D-Sensoren aufgeführt, die im unteren Preisbereich (unter 300 Euro) und auch für Privatpersonen erhältlich sind. Dabei ist für jede der eben vorgestellten 3D-Sensor-Technologien ein Beispielsensor aufgelistet. Vergleicht man den Kinect V1 Sensor aus dem Jahr 2010 mit dem Sensor Real Sense D435 von Intel aus dem Jahr 2018, ist der Real Sense Sensor in allen Bereichen deutlich überlegen. Sowohl die Reichweite, als auch die Auflösung und der Sichtbereich des Tiefensensors sind stark verbessert worden. Dabei ist der Intel Real Sense Sensor immer noch preiswert zu erwerben (180€).

Tabelle 3.1: Überblick über 3D-Sensoren und den Spezifikationen des enthaltenen Tiefensensors [AFG14; Asu18; Int18]. ASV = Active Stereo Vision, SL = Structured Light, ToF = Time of Flight.

Sensor	Auflösung	FPS	Reichweite	FOV H x V	Typ	Preis
Kinect V1 (2010)	320x240	30	1.8-3.5m	57° x 43°	SL	160€
Asus XtionPro (2011)	640x480	30	0.8-3.5m	58° x 45°	SL	140€
Kinect V2 (2014)	512x424	30	1.3-4.5m	70° x 60°	ToF	160€
Intel Real Sense D435 (2018)	1280x720	90	0.2-10m	91.2° x 65.5°	ASV	180€

3.3 Handsegmentierung

Die zuverlässige Lokalisierung der Hand spielt eine essenzielle Rolle bei allen Handerkennungsalgorithmen. Kann die Hand im Bild nicht lokalisiert werden, ist keine Bestimmung der Gelenkpositionen durchzuführen. Im Folgenden werden Handsegmentierungsverfahren vorgestellt, welche möglichst wenige strikte Anforderungen an das Eingabebild stellen und trotzdem verlässliche Resultate erzielen können. Verfahren, die zu viele strikte Einschränkungen, wie beispielsweise ein statischer Hintergrund oder die Voraussetzung, dass Hände das einzige Objekt im Bild sind, an die aufgenommene Szene stellen, werden nicht betrachtet.

3.3.1 Farbbasierte Verfahren

Die Lokalisierung der Hand anhand der Hautfarbe, ist einer der ersten Methoden, welche für die Handsegmentierung verwendet worden ist. Hierfür werden im Bild alle Pixel, die keinen Hautfarbton besitzen geschwärzt oder entfernt. Bei der Benutzung dieses Verfahrens treten Hindernisse auf. Beispielsweise stören zusätzliche Objekte im Bild, die ebenfalls einen Hautfarbton besitzen. Dadurch kann es zu Fehlerkennungen kommen, da die Hand nicht mehr anhand des Farbtons von den anderen Objekten unterschieden werden kann. Weiterhin können unterschiedliche Lichtverhältnisse zu Fehlerkennungen führen. Ebenfalls zu beachten sind die vielen unterschiedlichen Hauttöne, da der Hautfarbton sich von Mensch zu Mensch stark unterscheiden kann. Das Farbspektrum, das hier abgedeckt werden muss, ist nicht zu vernachlässigen [SSN+14]. Ein Ansatz um diese Probleme zu umgehen, ist die Verwendung von Farbräumen wie HSV oder YUV. Auf diese Weise kann der Einfluss von Beleuchtungs- und Lichtverhältnissen größtenteils aufgelöst werden. Zusätzlich wird in vielen Methoden ein vorher definiertes Farbmodell verwendet. Dieses ist bereits anhand einer großen Datenmenge an Bildern abgeleitet worden. Damit kann für jeden Pixel bestimmt werden, ob er einen Hautton besitzt oder nicht [ZBA09].

Trotz dieser Verbesserungen ist die Segmentierung der Hand durch Farbeigenschaften fehleranfällig und nicht zuverlässig genug für viele Anwendungsfälle. Deswegen werden inzwischen andere Verfahren oder Kombinationen mit anderen Methoden eingesetzt. Tang [Tan11] verknüpft das farbbasierte Verfahren mit Tiefen-Thresholding. Zuerst werden alle Pixel als Haut-Pixel oder Nicht-Haut-Pixel klassifiziert und dann die Tiefenwerte dieser Pixel in Kontext genommen. Befindet sich beispielsweise eine Haut-Pixel Region ganz am hinteren Rand der aufgenommen Szene kann diese Region ausgeschlossen werden. Auf diese Weise konnten Fehlerquellen wie andere Objekte mit Hautfarbton, die sich im Hintergrund befinden, entfernt werden.

3.3.2 Tiefen-Thresholding

Mit der Einführung der 3D-Sensorik kann Tiefen-Thresholding für die Handsegmentierung eingesetzt werden. Die meisten 3D-Sensoren können neben dem Tiefenbild zusätzlich ein RGB-Farbbild aufnehmen. Auf diese Weise werden beide Bildinformationen genutzt. Für jeden Pixel im Bild existiert ein Tiefenwert, welcher angibt, wie weit dieser Pixel von der Kamera entfernt ist. Dadurch lassen sich bestimmte Regionen im Bild extrahieren. Mo und Neumann [MN06] stellt den 3D-Sensor ungefähr 1 Meter vom Benutzer entfernt auf. Hiermit konnte davon ausgegangen werden, dass die Pixel, welche die Hand beschreiben, diejenigen sind, welche den kleinsten Tiefenwert besitzen.

Durch Tiefen-Thresholding kann mit hoher Genauigkeit die Hand im Bild segmentiert werden. Dies ist darauf beschränkt, dass sich keine Objekte vor den Händen befinden und dass die Kamera direkt vor dem Nutzer aufgestellt ist.

Um das Tiefen-Thresholding robuster zu machen und weniger Einschränkungen vorzugeben, wurde es mit zusätzlichen Verfahren erweitert beziehungsweise kombiniert [SM12]. Li und Jarvis [LJ09] schreibt eine Mindestanzahl an Pixel vor, aus der die Hand bestehen soll. Auf diese Weise werden kleine Objekte, die sich vor der Hand befinden, nicht als mögliche Handregionen vorgeschlagen. Cerlinca und Pentiuc [CP11] verwendet eine Gesichtserkennung, um das Gesicht der Person im Bild zu erkennen. Daraufhin kann ein passender Threshold für die Hand-Region im Bild berechnet werden. Auch diese Methode hat Nachteile. Es wird vorausgesetzt, dass sich ein Gesicht im Bild befindet und es kann es zu Problemen führen, wenn sich das Gesicht einer zweiten Person im Bild befindet.

3.3.3 Verfahren mit maschinellem Lernen

Erste Methoden für die Segmentierung der Hand im Bereich maschinellem Lernen basieren auf Entscheidungsbäumen. Shotton et al. [SFC+11] verwendet Random-Forests, um die Körperbereiche pixelweise zu segmentieren. Der Random-Forest gibt für jeden Pixel einen Wahrscheinlichkeitswert an, ob er zur einem Körperteil gehört oder zum Hintergrund. Inzwischen verwenden die meisten Verfahren neuronale Netze. Die erste Architektur, die gute Ergebnisse erzielte, ist das Region Based Convolutional Neural Network (R-CNN). Hier werden mögliche Regionen, in denen sich das Objekt befinden könnte, vorgeschlagen. Diese werden an das R-CNN weitergegeben. Es wird für jede vorgeschlagene Region ein Wert zurückgegeben, der angibt mit welcher Wahrscheinlichkeit sich das Objekt darin befindet. Da jede vorgeschlagene Region durch das R-CNN geschickt wird und keine der Berechnungen geteilt werden, ist dieses Verfahren langsam [RHGS15]. Deswegen sind weitere Methoden wie das Fast-R-CNN, Faster-R-CNN, YOLO und SSD entwickelt worden. Diese verwenden effizientere Verfahren für die Berechnung der vorgeschlagenen Regionen, wie auch für den Durchlauf durch das CNN. Huang et al. [HRS+17] führt eine Beschreibung und Vergleich dieser Methoden durch. Allgemein gilt, dass schnellere Methoden wie YOLO in Echtzeit laufen, dafür aber die Genauigkeit der Erkennung sinkt. Es gibt einen Tradeoff zwischen Geschwindigkeit und Genauigkeit.

Auch diese Methoden haben Nachteile. Es kommt vor, dass vorgeschlagene Regionen nicht die ganze Hand umfassen oder es gibt Fehlerkennungen. Für das Erkennen in Echtzeit leidet oft die Genauigkeit. Ein Vorteil dieser Methoden ist, dass keinerlei Anforderungen an das Eingabebild gestellt werden und sie in jedem Szenario eingesetzt werden können.

3.4 Handposendatensätze

Es existieren bereits Handposendatensätze, an denen die verschiedenen Methoden getestet und trainiert werden können. In Tabelle 3.2 ist eine Übersicht aktueller Handposendatenbanken dargestellt. Beispielbilder solcher Datensätze sind in Abbildung 3.5 aufgeführt. Grundsätzlich lässt sich zwischen synthetisch erzeugten Bildern und aufgenommen Bildern unterscheiden. Die aufgenommenen

Bilder sind mit 3D-Sensorik erstellt worden. Alle der untersuchten Datensätze für die Handerkennungsalgorithmen haben einen Tiefensensor verwendet. Synthetische Bilder werden mithilfe eines selbst erstellten Handmodells erzeugt. Ein Beispiel stellt die MSRC Datenbank von Microsoft da [SKR+15]. Hier werden 102000 synthetische Bilder erzeugt, um den Handerkennungsalgorithmus zu trainieren. Beispielbilder hieraus sind in Abbildung 3.5 dargestellt.

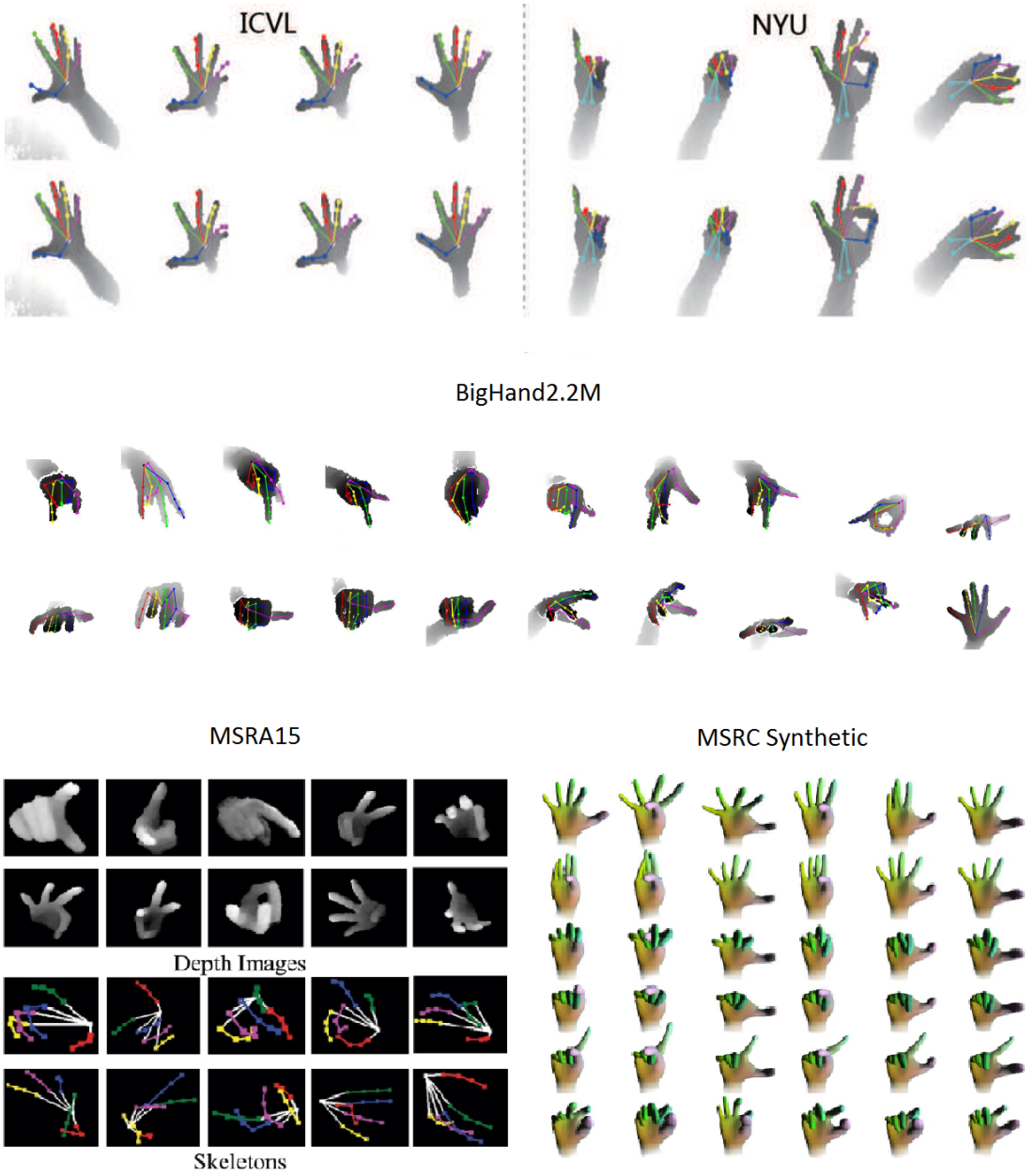


Abbildung 3.5: Darstellung möglicher Ergebnisse der Handposenerkennung anhand des NYU, ICVL, MSRA15 und MSRC Datensatz [GWC+17; HZX+18; SKR+15; YYS+17].

Für jedes der Bilder in den Datensätzen existieren die zugehörigen Gelenkkoordinaten. Dabei unterscheiden sich die Datensätze deutlich in der verwendeten Anzahl an Gelenkpunkten. Die Spanne reicht von fünf bis zu 36 annotierten Gelenkpunkten. Dafür wird meist ein Gelenkpunkt für das Handgelenk und die restlichen Gelenkpunkte für die Finger verwendet. Ein Beispiel für die Darstellung von 16 Gelenkpunkten, wie in dem ICVL Datensatz verwendet, ist in Abbildung 3.6 aufgelistet.

Die Perspektive, in welcher die Bilder aufgenommen werden, ist ein weiteres Merkmal der Datensätze. Hierbei ist zwischen 3rd-Person und der Ego-Perspektive zu unterscheiden. Die Ego-Perspektive beschreibt eine Sicht, welche dem menschlichem Sichtfeld nahe kommt, beispielsweise wird eine solche Perspektive durch eine am Kopf montierte Kamera erreicht. Die 3rd-Person Perspektive beschreibt eine Sichtweise die entsteht, wenn zum Beispiel eine Person eine andere mit einer Kamera aufnimmt. Die Kamera kann dabei auch an einer Halterung montiert werden. Die BigHand2.2M Datenbank ist dar einzige Datensatz, der beide Perspektiven verwendet und ist gleichzeitig der Datensatz mit der größten Anzahl an Bildern (2.2 Millionen). Dabei ist zu beachten, dass die Bilder in drei verschiedene Bereiche aufgeteilt werden: Handerkennung, Handtracking und Hand-Objekt-Interaktion. Ein weiterer Unterschied zwischen den Datensätzen, ist die Anzahl der verschiedenen Testpersonen, welche für die Aufnahmen eingesetzt werden. Hierbei reicht die Spanne von einer Testperson (Dexter 1, MSRC), bis zu 30 verschiedenen Testpersonen (ASTAR). Die Auflösung der verwendeten Tiefenbilder beträgt größtenteils 320x240 Pixeln. Neuere Datenbanken wie NYU oder BigHand verwenden inzwischen eine Auflösung von 640x480 Pixeln [YYS+17].

Die meisten aktuellen Methoden evaluieren ihre Verfahren auf dem ICVL-, NYU- und MSRA15 Datensatz. Der ICVL Datensatz besteht aus 17604 Tiefenbildern mit einer Auflösung von 320x240 Pixeln. Im Vergleich mit den anderen , ist er der kleinste Datensatz und beinhaltet größtenteils simple Handposen,die mit 16 Gelenkpunkten annotiert sind. Der MSRA15 Datensatz beinhaltet 76375 Tiefenbilder mit einer Auflösung von 320x240 Pixeln und 21 annotierten Gelenkpunkten und ist deutlich größer als der ICVL Datensatz. Er deckt im Vergleich ein wesentlich größeres Spektrum an Handposen ab. Den herausforderndsten Datensatz bei ICVL, NYU und MSRA15 stellt der NYU Datensatz dar. Er besteht aus 81009 Tiefenbildern mit einer Auflösung von 640x480 Pixeln und 36 annotierten Gelenkpunkten [MCL17]. Im Folgenden werden die verschiedenen Verfahren, sofern ausgewertet, auf dem ICVL Datensatz verglichen, da dieser am häufigsten zur Evaluation eingesetzt worden ist.

3.5 Modellbasierte Methoden

Modellbasierte Methoden sind alle Verfahren, welche ein vorher definiertes Handmodell verwenden. Dafür wird ein kinematisches Handmodell erstellt, sowie meist ein zusätzliches Modell für die Handoberfläche. Ein Beispiel hierfür ist in Abbildung 3.6 dargestellt. Das Handmodell wird durch dessen Gelenkpunkte, deren Freiheitsgrade und den Rotationswinkeln beschrieben. Auf diese Weise können Einschränkungen auf mögliche Handposen gelegt werden, das heißt Handposen, welche physisch vom Menschen nicht ausführbar sind, können ausgeschlossen und der Suchraum an möglichen Posen eingegrenzt werden.

Tabelle 3.2: Übersicht von verschiedenen Handposendatensätzen [YYS+17]

Datensatz	Art	Bilder	Gelenkpunkte	Subjekte	Perspektive	Auflösung
Dexter 1	Tiefenbilder	2137	5	1	3rd	320x240
MSRA14	Tiefenbilder	2400	21	6	3rd	320x240
ICVL	Tiefenbilder	17604	16	10	3rd	320x240
NYU	Tiefenbilder	81009	36	2	3rd	640x480
MSRA15	Tiefenbilder	76375	21	9	3rd	320x240
UCI-EGO	Tiefenbilder	400	26	2	ego	320x240
Graz16	Tiefenbilder	2166	21	6	ego	320x240
ASTAR	Tiefenbilder	870	20	30	3rd	320x240
HandNet	Tiefenbilder	212928	6	10	3rd	320x240
MSRC	Synthetisch	102000	22	1	3rd	320x240
BigHand2.2M	Tiefenbilder	2200000	21	10	komplett	640x480

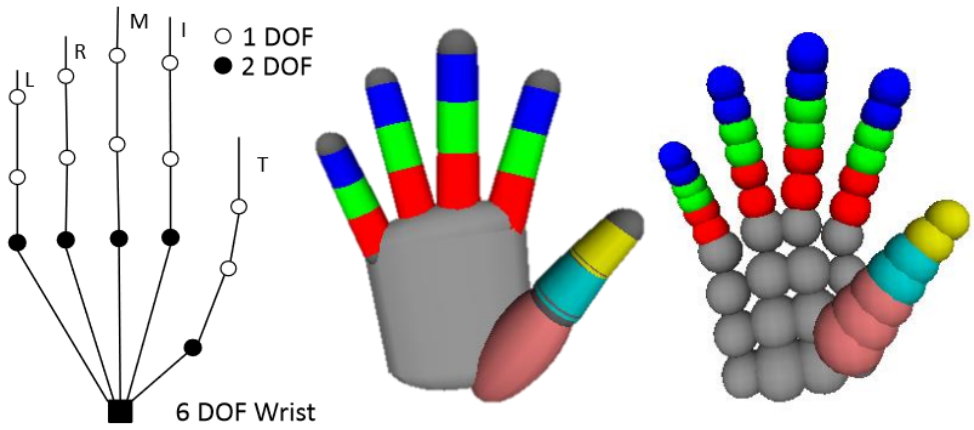


Abbildung 3.6: Darstellung eines Handmodells mit 16 Gelenkpunkten und 26 Freiheitsgraden und möglichen Handoberflächenmodellen [QSW+14].

Die Herangehensweise der modellbasierten Methoden ist folgendermaßen aufgebaut. Zunächst wird eine Funktion festgelegt, welche die Unterschiede zwischen dem Eingabebild und dem vorher definierten Handmodell beschreibt. Hierfür wird eine Fehlerfunktion verwendet, die diese Differenz angibt. Um den Fehler zu minimieren wird ein Optimierungsalgorithmus basierend auf der Fehlerfunktion definiert. Dieser gibt die synthetische Handpose des Handmodells zurück, die den kleinsten Fehler im Bezug zu dem beobachteten Eingabebild aufweist. Solche Optimierungsalgorithmen benötigen eine Initialisierung, oft wird dafür die Handpose verwendet, welche im vorherigen Durchlauf erkannt worden ist. Bei einer schlechten Initialisierung kann der Optimierungsalgorithmus in lokalen Minima stecken bleiben und aufgrund dessen starke Abweichungen von der realen Handpose entstehen. Ein weiterer Nachteil von modellbasierten Verfahren ist, dass für jeden Nutzer der Methode das Handmodell auf die eigene Hand angepasst werden muss. Ein Handmodell kann nicht generell für alle Personen verwendet werden, da Hände in Größe und Form von Mensch zu Mensch stark voneinander abweichen [EBN+07; LFP11].

Die verschiedenen modellbasierten Verfahren unterscheiden sich bei dem verwendeten Handmodell, der Fehlerfunktion und der gewählten Optimierungsfunktion. In [OKA11] wird als Optimierungsfunktion Particle Swarm Optimisation (PSO) verwendet. Zunächst wird die Hand im Bild basierend auf Farbe und Tiefeninformationen segmentiert. Das resultierende Bild ist ein Tiefenbild der Hand. Dies wird mit dem gerenderten Bild des verwendeten Handmodells verglichen. PSO ist ein stochastischer Algorithmus, der die Optimierung einer Funktion durch die Bewegung von Partikeln, die sich im Parameterraum der zu optimierenden Funktion befinden, beschreibt. Jedes Partikel speichert seine aktuelle Position und Geschwindigkeit. Diese Partikel werden durch den Suchraum bewegt, um die beste Lösung zu finden, welche in diesem Fall die am besten passende Handpose darstellt. Eine detaillierte Beschreibung der Implementierung des PSO Algorithmus ist in [OKA11] zu finden. Für die Initialisierung bei PSO wird die zuletzt gefundene Handpose benutzt und zur Evaluation der Methode ein eigens erstellter Datensatz mit dem Kinect Sensor verwendet. Auf 900 Testbildern, ist eine Genauigkeit von 74 Prozent erreicht worden. Dabei liegt die erlaubte Abweichung aller Gelenkkoordinaten gesamt bei maximal 40 mm.

Liang et al. [LYT12] verwendet im Gegensatz zu Oikonomidis et al. [OKA11] die Optimierungsfunktion ICP. Sowohl mit dem Eingabebild, als auch der synthetische Handmodellpose wird eine Punktwolke generiert. Beide Punktwolken dienen als Input für den ICP Algorithmus, dieser beschreibt eine Optimierungsfunktion, welche eine Punktwolke der anderen annähert, sodass der Abstand der einzelnen Punkte beider Wolken möglichst gering ist. Dieses Verfahren wird iterativ wiederholt. In Abbildung 3.7 ist ein Beispiel für den Durchlauf des ICP Algorithmus abgebildet. Eine detaillierte Beschreibung der ICP Methode ist in [MSFM10] zu finden. Die Initialisierung der Handpose für den ICP Algorithmus wird mithilfe einer Fingerspitzenerkennung verfeinert. Anhand der Positionen der Fingerspitzen wird eine initiale Handpose mithilfe des synthetischen Handmodells generiert. Der ICP Algorithmus errechnet darauf basierend die finale Handpose. Für die Evaluation dient ein mit dem Kinect Sensor selbst erstellter Datensatz. Allerdings sind keine tatsächlichen Wahrheitswerte für die Gelenkpositionen ermittelt worden, daher ist keine Auswertung im Bezug auf die Genauigkeit der bestimmten Gelenkpositionen vorhanden.

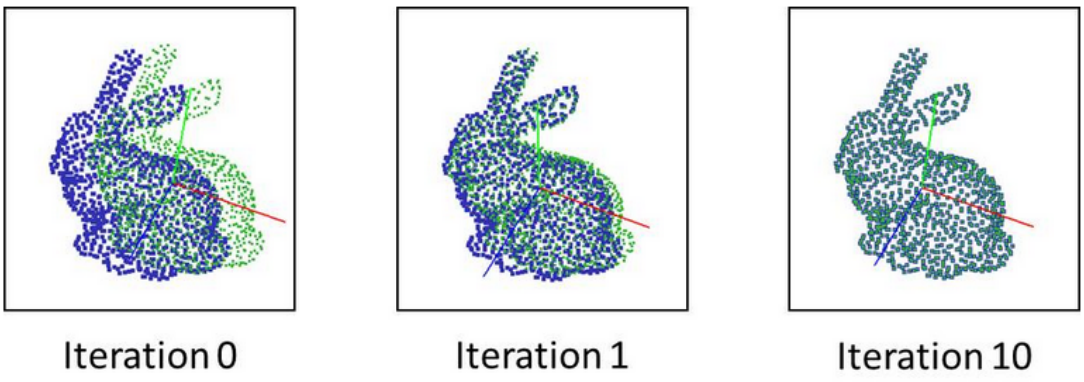


Abbildung 3.7: Durchlauf des ICP Algorithmus an zwei Punktwolken [san].

Qian et al. [QSW+14] verwendet für die Optimierung eine Kombination aus ICP und PSO. Bei einer initialen Pose mit einer Abweichung von maximal 15-25 mm bei den Gelenkpunkten zu dem realem Warheitswert, liegt die Genauigkeit bei der ICP-PSO Methode bei 90.8 Prozent. Die PSO Methode für sich alleine erreicht eine Genauigkeit von 52.4 Prozent, ICP erreicht 61.4 Prozent. Durch die

Kombination von ICP und PSO kann die Genauigkeit gegenüber der getrennten Verwendung der Methoden fast um das zweifache gesteigert werden. Für die Auswertung ist ein selbst erstellter Datensatz verwendet worden.

Zhou et al. [ZWZ+16] verwendet einen anderen Ansatz. Das Handmodell ist innerhalb eines CNN integriert. Der Aufbau des Netzwerks ist in Abbildung 3.8 dargestellt. Das Besondere ist das Handmodell Layer, welches in das CNN integriert wird. Es mappt die extrahierten Handparameter aus dem Eingabebild zu den Handgelenkpunkten des verwendeten Handmodells. Hierfür wird eine kinematische Funktion verwendet und somit kann eine korrekte physisch ausführbare Handpose als Ergebnis garantiert werden. Das CNN wird in einem Durchgang trainiert und gibt direkt die errechneten Gelenkpositionen zurück. Die Auswertung der Methode ist auf mehreren öffentlich verfügbaren Datensätzen durchgeführt worden. Auf dem ICVL Datensatz, mit einer maximalen erlaubten gesamten Gelenkpositionsabweichung von 40 mm, wird eine Genauigkeit von ca. 84 Prozent erreicht.

Alle Methoden, die hier verglichen werden, verwenden unterschiedliche Datensätze zur Evaluierung. Deshalb können die Ergebnisse untereinander nicht wirklich in Relation gesetzt werden. Da das hier zuletzt vorgestellte Verfahren auf öffentlich zugänglichen Datensätzen ausgewertet worden ist, lässt sich dieses mit den folgenden neueren Ansätzen und Methoden vergleichen.

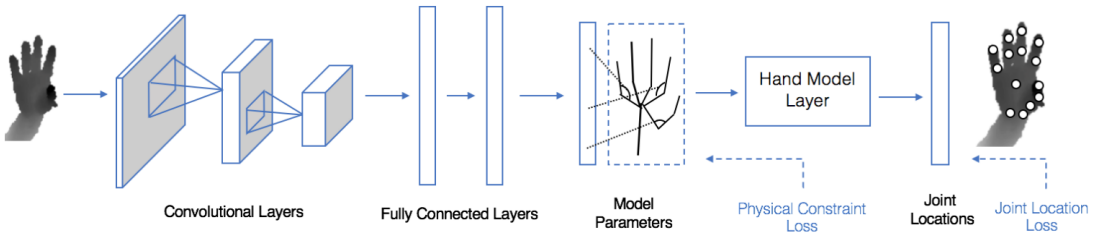


Abbildung 3.8: Aufbau eines CNN für einen modellbasierten Handposenerkennungsalgorithmus. In den letzten Layern wird ein Handmodelllayer integriert um die Einschränkungen durch das kinematische Handmodell in das Netzwerk zu integrieren [ZWZ+16].

3.6 Datenbasierte Methoden

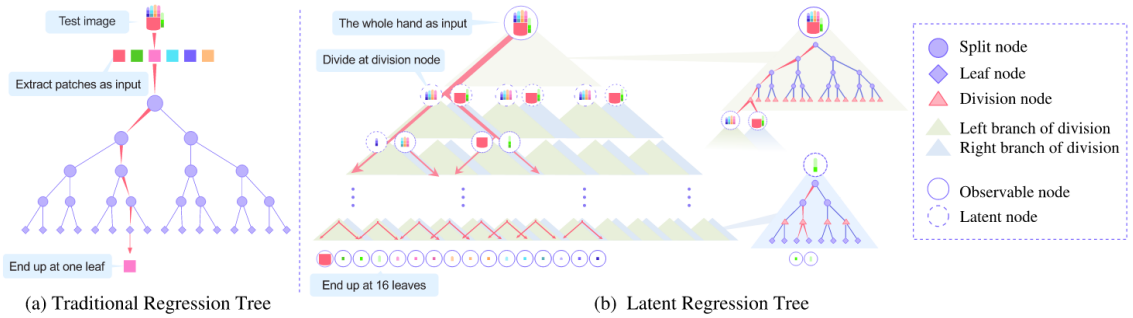
Datenbasierte Methoden können direkt anhand des Eingabebildes Aussagen über die Handpose treffen. Um dies zu erreichen, wird im Allgemeinen folgende Herangehensweise verwendet. Es existiert ein Datensatz, welcher Bilder von verschiedenen Handposen enthält. Für jedes dieser Bilder müssen die realen Gelenkpositionen der Hand angegeben sein. Mithilfe eines solchen gelabelten Datensatzes werden Verfahren mit maschinellem Lernen trainiert und ermöglichen anhand eines beliebigen Handbildes die Gelenkpositionen ableiten zu können. Wie ein solches Ergebnis aussehen kann, ist in Abbildung 3.5 dargestellt. Die Verwendung von datenbasierten Methoden bringt Herausforderungen mit sich. Es ist mit einem großen Aufwand verbunden den benötigten gelabelten Datensatz, welcher eine genügend großen Anzahl an qualitativ hochwertigen Bildern enthält, anzulegen. Zusätzlich fordern die meisten Ansätze zusätzliche Zeit für das Trainieren der Verfahren. Ein weiterer Nachteil liegt darin, dass meist keine Einschränkungen auf die vorhergesagten

Gelenkpunkte gemacht werden, wie es bei allen modellbasierten Methoden der Fall ist. Dadurch können Handposen vorhergesagt werden, welche rein physisch gesehen für einen Menschen nicht ausführbar sind [EBN+07; MZ15].

Einer der ersten Algorithmen aus dem Bereich des maschinellem Lernens, welche für datenbasierte Verfahren eingesetzt worden, sind Entscheidungsbäume (Random Forest). Xu und Cheng [XC13] trainieren zwei Random-Forests mithilfe eines synthetisch erzeugten Handmodells. Der erste Random-Forest ist dafür zuständig die 3D-Position und Rotation der Hand im Bild zu bestimmen. Das Ergebnis dient als Input für den zweiten Random-Forest, welcher die Form und Position der restlichen Punkte des verwendeten Handmodells bestimmt. Für das Training wird eine große Anzahl an synthetischen Bildern verwendet, aufgrund dessen müssen die realen Testbilder erst aufgearbeitet werden. Hierfür wird ein initiales Bild der Testhand benötigt um diese an die Größe des verwendeten synthetischen Modells anzupassen. Da die Berechnungen bei Random Forests effizient und ressourcenarm durchzuführen sind, erreicht diese Methode bereits zu der Zeit (2013) eine Laufzeit von 67 FPS mit der Verwendung einer GPU. Als Ergebnis wird ein synthetisch erzeugtes Handmodell zurückgegeben. In Abbildung 3.9 sind mehrere Beispielergebnisse dieses Verfahren dargestellt.

Abbildung 3.9: Beispielergebnisse für die Random-Forest Methode. Die erste Reihe stellt die Eingabebilder und die zweite Reihe die Resultate dar [XC13].

Weitere ähnliche Methoden, die auf Random-Forests basieren, sind in [KKA14; KKKA12; TYK13] zu finden. Tang et al. [TJTK14] verfolgt einen neueren Ansatz basierend auf Random Forestss. Im Gegensatz zu den vorherigen Methoden, welche jeden Pixel einzeln klassifizieren und darauf aufbauend die Gelenkkoordinaten ableiten, wird hier die gesamte Hand als Input genommen. Das Verfahren kann als von grob zu fein strukturierte Suche beschrieben werden, welches beim Zentrum der Hand anfängt und die Suche weiter in kleine Bestandteile aufgliedert bis alle Gelenkpunkte gefunden wurden. Ein Vergleich beider Ansätze ist in Abbildung 3.10 dargestellt.



(a) Traditional Regression Tree (b) Latent Regression Tree

Abbildung 3.10: Vergleich des Latent Regression Forest Modell gegenüber eines herkömmlichen Random-Forest Modell. [TJTK14].

Als Eingabe dient die gesamte Punktwolke der Hand und dessen Massenmittelpunkt (Center of Mass). Hierfür wird ein Latent Tree Model verwendet, welches aus mehreren Latent Regression Forests besteht (Entscheidungsbäume), die die Struktur der Hand abbilden sollen. Ein Beispiel für die Bestimmung eines einzelnen Gelenkpunkts ist in Abbildung 3.11 dargestellt. Die Hand wird in kleine Teilregionen aufgeteilt bis am Ende der Kette der Gelenkpunkt klassifiziert worden ist.

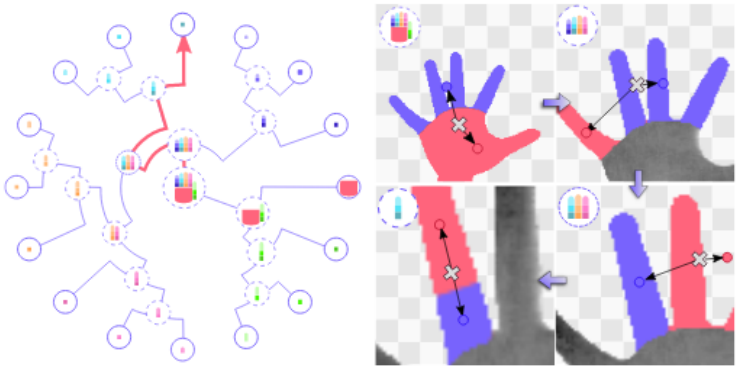


Abbildung 3.11: Beispieldurchlauf des Latent Regression Forest für einen einzelnen Gelenkpunkt. Die Hand wird in kleinere Regionen aufgeteilt bis am Ende der Kette Gelenkpunkt gefunden wird [TJTK14].

Eine genaue Beschreibung eines Latent Tree Model ist in [CTAW11] erläutert. Durch die Verwendung dieses Modells konnte die Genauigkeit der Erkennung erhöht werden. Bei einem maximalen Gelenkpositionabweichungsfehler von 40 mm erzielte dieser Ansatz eine Genauigkeit von ca. 63 Prozent auf dem ICL Testdatensatz. Normale Random-Forest Methoden wie [KKKA12] erreichten bei denselben Testbedingungen eine Genauigkeit von ca. 22 Prozent [SRY+15; TJTK14]. Die erzielten Ergebnisse sind gegenüber den herkömmlichen Random-Forest Methoden durch die Verwendung des Latent Tree Modells fast um ein dreifaches verbessert worden. Betrachtet man das Ergebnis von 63 Prozent bei einer erlaubten Gelenkspositionsabweichung von 40 mm, ist dieses allerdings weit entfernt von einer genauen und zuverlässigen Erkennung. Bereits simple auf CNN basierenden Methoden wie DeepPrior (welches in den folgenden Absätzen genauer beschrieben wird), erzielen hier deutlich bessere Ergebnisse. Im Vergleich dazu erreichte DeepPrior bei den eben genannten Testumständen eine Genauigkeit von ca. 90 Prozent. Dies ist eine deutliche Steigerung. Deswegen werden inzwischen im Bereich der datenbasierten Verfahren fast ausschließlich Methoden verwendet die auf neuronalen Netzen aufbauen. Im Folgenden werden Beispiele für solche Verfahren näher vorgestellt.

Eines der ersten Verfahren, welches ein CNN für die Handposenerkennung verwendet, ist DeepPrior. Oberweger et al. [OWL15] vergleicht hier tiefe-, flache- und Netzwerkarchitekturen, die mehrere Auflösungen unterstützen, miteinander. Als Input für das jeweilige CNN dient ein auf 128x128 skaliertes Tiefenbild der Hand. Dessen Tiefenwerte sind auf den Bereich [-1,1] normalisiert. Für die Segmentierung der Hand wird angenommen, dass sie das nächste Objekt zum verwendeten Sensor ist. Bei dem Vergleich der Netzwerkarchitekturen ergibt sich, dass tiefe Architekturen, welche zusätzlich mehrere Auflösungen unterstützen, die besten Ergebnisse erzielen, während flache Architekturen am schlechtesten abschneiden. Anstatt die 3D-Koordinaten der Gelenke direkt aus dem Eingabebild abzuleiten, werden diese Handposenparameter in einen kleiner dimensionierten

Parameterbereich abgebildet. Hierfür wird ein sogenanntes ”Bottleneck Layer” vor dem letztem Layer eingesetzt. Dieses zwingt das Netzwerk eine niedrig dimensionierte Repräsentation der Trainingsdaten zu lernen, welche die physischen Einschränkungen der Hand, die ebenfalls den Raum möglicher Handposen begrenzt, widerspiegeln soll. Dies wird erreicht indem ein Anteil der Neuronenverbindungen für dieses Layer entfernt werden. Da dieser Vorgang durch auf bereits zuvor bekannten Informationen basiert (die möglichen Handposen sind bekannt), wird das zusätzlich eingeführte Bottleneck Layer hier als "Prior"bezeichnet, was zu dem Namen DeepPrior der Methode geführt hat. Im letzten Layer wird diese niedrig dimensionierte Repräsentation wieder auf die volle ursprüngliche Dimension zurückgerechnet. Hierfür werden die Gewichte des Layers, welches die Parameter auf ursprüngliche Dimension zurück projiziert, mithilfe von der Principal Component Analysis auf Basis des verwendeten Datensatzes initialisiert. In Abbildung 3.12 ist die Architektur des Netzwerks mit einem solchen "Bottleneck Layer"dargestellt. Durch die Verwendung dieses kann die Anzahl der möglichen Handposen reduziert und deshalb die Zuverlässigkeit der Vorhersagungen erhöht werden. Die Genauigkeit des Verfahrens auf dem ICVL Datensatz (mit einer maximalen Abweichung von 20 mm bei den Gelenkpositionen) ist durch die Verwendung eines solchen Bottleneck Layer von 40 Prozent auf 45 Prozent erhöht worden.

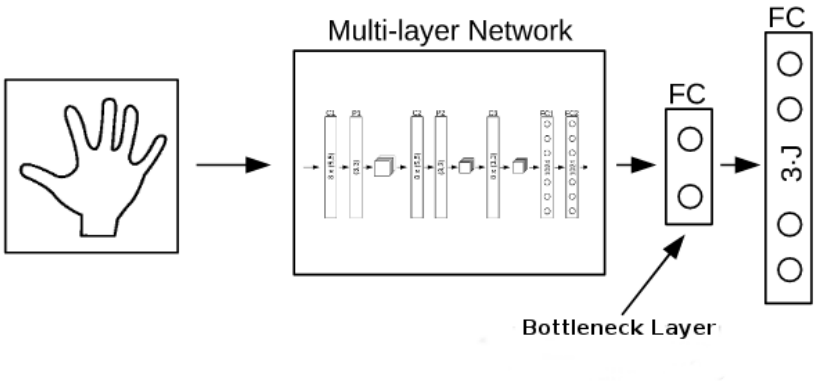


Abbildung 3.12: Netzwerkarchitektur mit einem Bottleneck Layer. Durch Einfügen dieses Layer wird das Netzwerk gezwungen eine niedrig dimensionierte Repräsentation der Trainingsdaten zu lernen [OWL15] .

Um die Genauigkeit des eben vorgestellten Verfahrens noch weiter zu erhöhen, wird eine iterative Methode nach dem Erhalt der Gelenkkoordinaten angehängt. Für jeden Gelenkpunkt werden weitere kleine Regionen an der Position der Gelenkkoordinaten extrahiert und durch ein weiteres dafür trainiertes Netzwerk geschickt. Dieses dient dazu die Genauigkeit der Bestimmung der Gelenkkoordinaten noch weiter zu verfeinern. Da es eine iterative Methode ist, werden die neu bestimmten Gelenkkoordinaten als Eingabe für den nächsten Durchgang verwendet. Die hier angewendeten Verbesserungen helfen die Genauigkeit zu erhöhen, jedoch nimmt die Trainingszeit zu und die Performance sinkt, die Methode läuft in Echtzeit bei der Berechnung auf einer fähigen GPU. Ohne den iterativen Verfeinerungsschritt lag die Laufzeit für einen Durchlauf bei 0.09 ms auf einer Geforce 780 Ti, mit der iterativen Verfeinerung bei 2.38 ms. Die gesamte Methode erreicht bei einer maximal erlaubten Gelenkpunktabweichung von 40 mm eine Genauigkeit von ca. 90 Prozent auf dem ICVL Testdatensatz.

Ein weiterer Ansatz ist die Verwendung von Multi-View-CNNs. Ge et al. [GLYT16] erstellt anhand eines Tiefenbilds eine 3D-Punktwolke. Diese wird auf drei verschiedene orthogonale Ebenen projiziert. Jedes dieser drei Bilder dient als Input für ein anderes CNN. Die drei Netzwerke generieren drei verschiedene 2D-Heatmaps. Hierbei wird durch die Intensität der Farbdarstellung ausgedrückt, welche 2D-Position am ehesten für den Gelenkpunkt in Frage kommt. Die Kombination dieser 2D-Heatmaps erlaubt am Ende die Berechnung der 3D-Koordinaten von den Gelenkpunkten. Die Verwendung mehrerer verschiedener Perspektiven sorgt für eine bessere Genauigkeit als die Verwendung von nur einer Perspektive. An dem Testdatensatz aus [SWL+15], kann durch die Verwendung mehrerer Perspektiven gegenüber einer, die Genauigkeit, mit einer maximal erlauben Gelenkpositionsabweichung von 40 mm, von ca. 54 Prozent auf ca. 80 Prozent erhöht werden. Die Methode ist in Abbildung 3.13 dargestellt. Da mehrere CNNs verwendet werden, erhöht sich die Trainingszeit wesentlich und die Performance sinkt. Die Methode läuft immer noch in Echtzeit mit der Verwendung einer fähigen GPU (hier Nvidia Tesla K20), bei einer Laufzeit von 14.1 ms für die Berechnung einer Handpose.

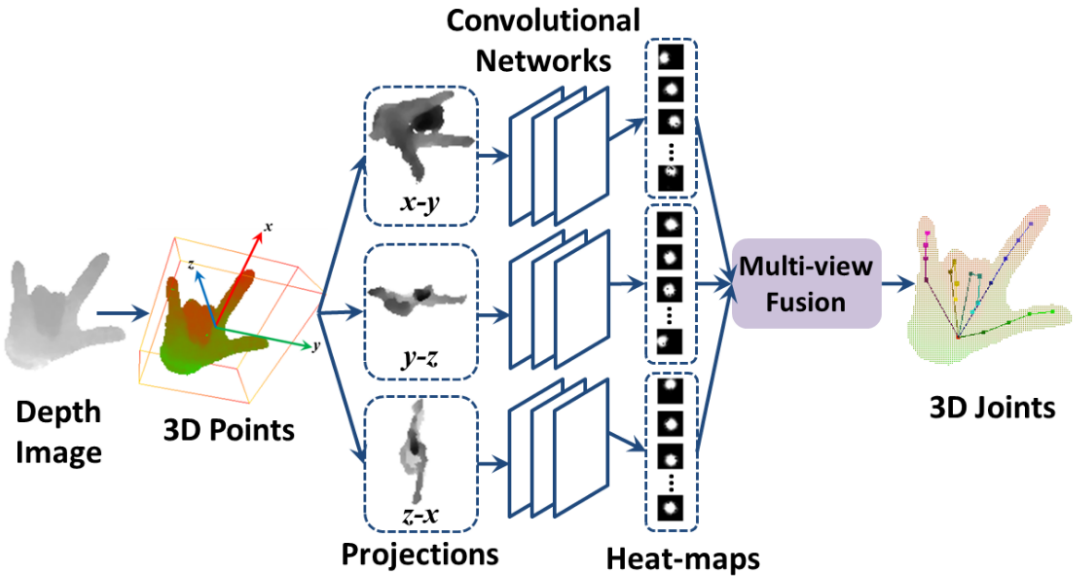


Abbildung 3.13: Netzwerkarchitektur des Multi-View-CNN Ansatzes. Das Eingabebild wird auf drei verschiedene Ebenen projiziert. Für jede Projektion werden Heatmaps erstellt, welche dann am Ende für die Berechnung der Gelenkkoordinaten wieder zusammengefügt werden. [GLYT16] .

Die Methode DeepPrior++ [OL17] baut auf der bereits vorgestellten Methode DeepPrior von [OWL15] auf. Als Verbesserung wird eine neuere Netzwerkarchitektur für die Bestimmung der Gelenkkoordinaten verwendet. Das Netz basiert auf dem ResNet [HZRS16], welches gute Ergebnisse in der Bildklassifizierung erzielt. Da es sich bei der Handposenerkennung um ein Regression-Problem handelt, werden Anpassungen vorgenommen. Das globale Pooling-Layer ist entfernt und zwei weitere FC-Layer hinzugefügt worden. Der Aufbau der neuen Netzwerkarchitektur ist in Abbildung 3.14 dargestellt. Da die ResNet-Architektur anfällig für Overfitting ist, sind weitere Dropout Layer zur Regulation in das Netzwerk integriert worden. Die Trainingsdaten werden

im Gegensatz zur ursprünglichen Methode zusätzlich mithilfe von Augmentation erweitert. Mit diesen Änderungen konnte die Genauigkeit auf dem ICVL Datensatz mit einer maximal erlauben Gelenkpunktabweichung von 40 mm von ca. 90 Prozent auf ca. 95 Prozent erhöht werden.

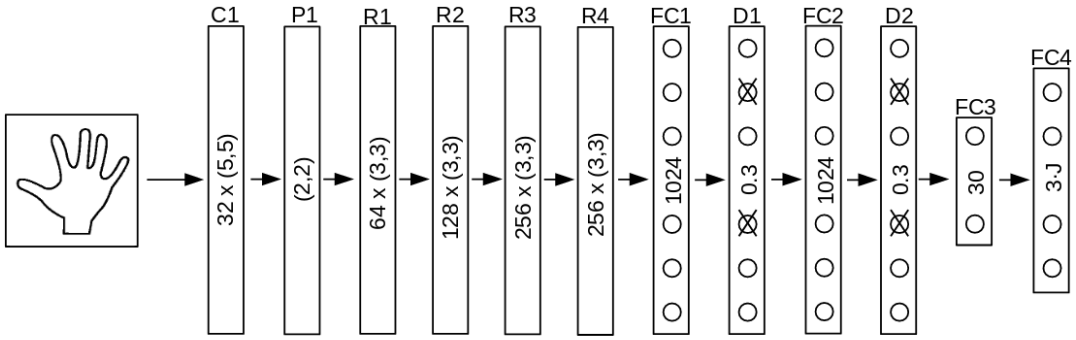


Abbildung 3.14: Netzwerkarchitektur von DeepPrior++ auf Basis eines ResNet. C beschreibt ein Convolutional-Layer, FC für ein FC-Layer, D ein Dropout-Layer, R ein Residual Modul und P ein Max-Pooling-Layer (MP-Layer) [OL17].

Ein weiterer moderner Ansatz kommt in [GWC+17] zum Einsatz. Die Netzwerkarchitektur basiert auf einem Region Ensemble Network. Der erste Teil des Netzwerks ist eine normale CNN-Architektur, welche die Feature Maps zurückgibt. Innerhalb dieser erhaltenen Features werden einzelne Regionen ausgewählt und jeweils durch ein weiteres FC-Layer gesendet. Am Ende werden alle Layer der verschiedenen Regionen zu einem einzelnen FC-Layer verbunden. Ein Überblick über die Methode ist in Abbildung 3.15 dargestellt. Das Netzwerk wird im Gegensatz zum Multi-View Ansatz ein einziges Mal trainiert, da die verwendeten Feature-Regionen nur einmal berechnet werden müssen. Dieses Verfahren erreicht auf dem ICVL Datensatz bei einer maximal erlaubten Gelenkpunktabweichung von 40 mm eine Genauigkeit von ca. 96 Prozent.

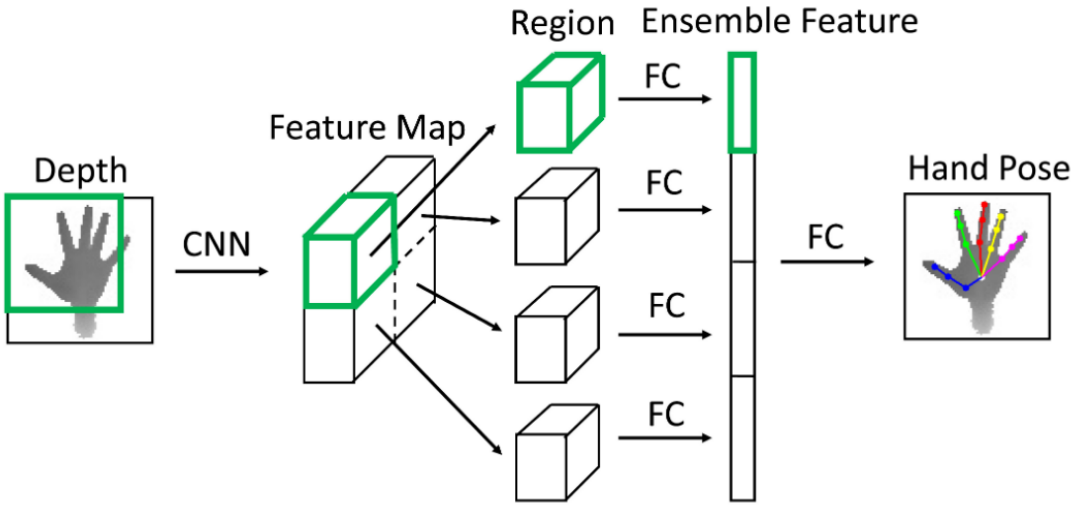


Abbildung 3.15: Netzwerkarchitektur des Region Ensemble Netzwerks [GWC+17].

[WPVY17] verwendet mit Dense Regression einen neuen Ansatz bei der Repräsentationen der Parameter. Anstatt mit absoluten Gelenkpositionen zu arbeiten, werden Offsets benutzt, das heißt jeder Pixel wird auf einen 3D-Offset zu jedem Gelenkpunkt zurückgeführt. Es werden sowohl die 2D- als auch die 3D- geometrischen Eigenschaften des Tiefenbildes wirksam genutzt. Von der 2D-Perspektive aus, wird das Tiefenbild als 2D-Oberfläche, welche in einem 3D-Raum liegt, betrachtet. Um geometrische Muster auf dieser Oberfläche einzufangen wird ein CNN benutzt. Von der 3D-Perspektive aus wird das Tiefenbild als ein Set an 3D-Punkten betrachtet. Für diese Punkte soll der Offset zu den Gelenkpunkten bestimmt werden. Hierfür wird ein CNN verwendet, welches ein Vektorfeld an Offsets für jeden Gelenkpunkt ableitet. Der Offset Vektor wird in zwei Komponenten zerlegt, eine 3D-Heatmap und einen direktionalen Einheitsvektor. Mithilfe der 2D-, 3D-Heatmap und dem Einheitsvektor wird die finale Handpose abgeleitet. Die Berechnung ist komplex und ist in [WPVY17] detailliert beschrieben. Der Aufbau der Netzwerkarchitektur ist in Abbildung 3.16 dargestellt. Als Grundgerüst wird die sogenannte Hourglass Netzwerkarchitektur, welche speziell für die Posenerkennung von Menschen entwickelt worden ist, verwendet. Eine genaue Beschreibung dieser ist in [NYD16] zu finden. Für jeden Gelenkpunkt wird eine 2D-Heatmap, eine 3D-Heatmap und ein Einheitsvektorfeld abgeleitet. Aus diesen Informationen können dann die 3D-Gelenkpositionen der Hand abgeleitet werden. Dieses Verfahren erreicht auf dem ICVL Datensatz eine Genauigkeit von ca. 97 Prozent, bei einer maximal erlaubten Gelenkpositionsabweichung von 40 mm.

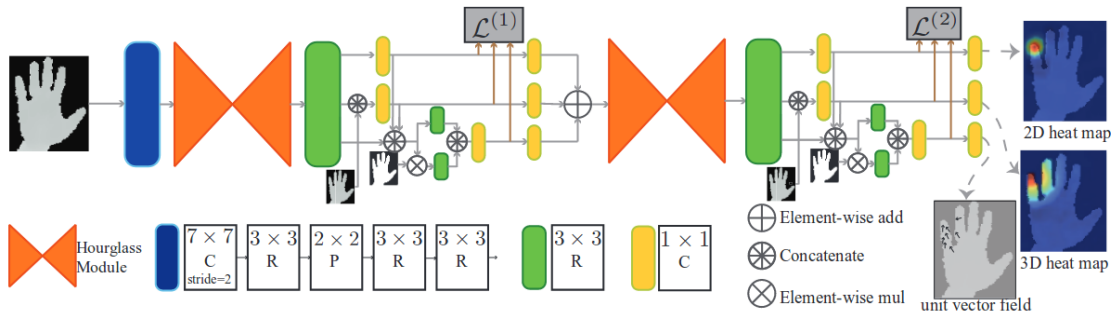


Abbildung 3.16: Netzwerkarchitektur des Dense Regression Netzwerks. Die Abkürzungen C,P,R stehen für Convolution Layer, Pooling-Layer und Residuale Module. Das Netzwerk bestimmt eine 2D-Heatmap, eine 3D-Heatmap und einen Einheitsvektorfeld für jeden Gelenkpunkt. In dieser Abbildung wird die Berechnung des Gelenkpunkts im kleinen Finger dargestellt [WPVY17].

Ein weiterer Ansatz ist die Verwendung von 3D-CNNs [GLYT17; MCL17], diese verwenden eine dritte zusätzliche Achse (x,y und z) für die Berechnung der Convolutions. In [MCL17] werden die Tiefenbilder in dreidimensionale volumetrische Formen umgewandelt. Hierzu werden die Punkte des Tiefenbilds in den 3D-Raum projiziert. Diese sogenannten Voxel dienen als Input für das Netzwerk. Es wird für jeden Voxel ein Wert zurückgegeben, welcher beschreibt, wie wahrscheinlich ein Gelenkpunkt enthalten ist. Die Position mit der höchsten Wahrscheinlichkeit für jeden Gelenkpunkt wird in die realen 3D-Koordinaten umgewandelt und zurückgegeben. Die Netzwerkarchitektur besteht aus einem Encoder, welcher für das Downsampling und einem Decoder der für das Upsampling verantwortlich ist. Zuerst wird der Input mithilfe des Encoders auf einer kleinere Feature Map reduziert und auf dieser werden dann für jeden Voxel der Wahrscheinlichkeitswert, welcher angibt wie wahrscheinlich es ein Gelenkpunkt ist, bestimmt. Danach werden die Featuremaps

durch den Decoder wieder auf ihre ursprüngliche Dimension umgewandelt. Die Netzwerkarchitektur basiert ebenfalls auf dem Hourglass Modell [NYD16], eine Übersicht ist in Abbildung 3.17 dargestellt. Aufgrund der Verwendung von Voxeln und einer komplexen Netzwerkarchitektur, ist die Berechnung aufwendig und die Methode benötigt eine starke GPU (hier wird ein Nvidia Titan X verwendet) um performant in Echtzeit laufen zu können. Dieses Verfahren hat die "Hands in the Million Challenge"[YYGK17] basierend auf 3D-Handposenerkennung im Jahr 2017 gewonnen und erzielt die besten Ergebnisse der hier verglichenen Methoden. Auf dem ICVL Datensatz ist bei einer Gelenkpositionsabweichung von 40 mm eine Genauigkeit von ca. 98 Prozent erreicht worden.

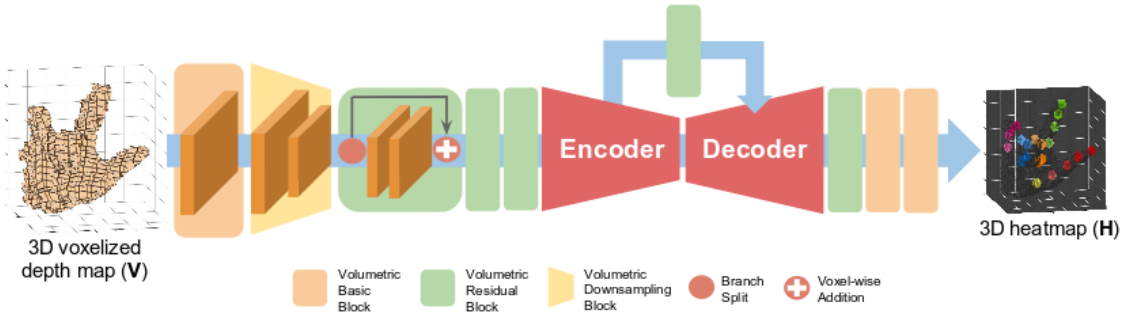


Abbildung 3.17: Netzwerkarchitektur des 3D-CNN mit Voxel. Für jeden Voxel wird ein Wahrscheinlichkeitswert berechnet, der angibt wie wahrscheinlich er ein Gelenkpunkt ist und am Ende wird darauf basierend eine 3D-Heatmap zurückgegeben. Jede Farbe in der Heatmap steht für Gelenkpunkte in demselben Finger [MCL17].

3.7 Hybride Methoden

Hybride Methoden verbinden datenbasierte und modellbasierte Verfahren miteinander. Dadurch sollen die Vorteile beider Methoden miteinander kombiniert werden. Da modellbasierte Methoden stark abhängig von einer guten Initialisierung sind, ist ein Ansatz der hybriden Methoden datenbasierte Verfahren für die Berechnung der zur Initialisierung verwendeten Handpose einzusetzen. Tompson et al. [TSLP14] ist einer der ersten, welcher diesen Ansatz verfolgt. Für die Handsegmentierung wird ein Random-Forest verwendet. Das segmentierte Bild dient als Input für ein CNN, welches die Gelenkpunkte in Form von Heatmaps zurückgibt. Diese dienen als Initialisierung für den modellbasierten Teil, welcher eine inverse kinematische Funktion verwendet. Auf diese Weise kann das Problem von stark abweichenden Initialisierungen verbessert werden. Einen ähnlichen Ansatz verwendet Sanchez-Riera et al. [SSH+17]. Hier wird ein simples CNN genutzt um die initiale Handpose zu bestimmen. Dann wird mithilfe des ICP Algorithmus und des Eingangbildes die finale Handpose bestimmt. Sharp et al. [SKR+15] gibt mithilfe einer datenbasierte Methode anhand eines Eingabebildes mehrere mögliche Handposen zurück. Der modellbasierte Teil des Verfahrens berechnet für jede dieser Posen die am besten passende Pose des synthetischen Handmodells. Am Ende wird die synthetische Handmodellpose gewählt, welche am nächsten an dem realem Eingabebild dran ist. Malik et al. [MES17] versucht das Problem der unterschiedlichen Handgrößen bei modellbasierten Verfahren zu lösen, indem das verwendete Handmodell automatisch für jeden Nutzer neu kalibriert wird. Bei diesem hybriden Verfahren ist das CNN nicht nur für die

Bestimmung der Gelenkpunkte als Initialisierung zuständig, sondern gibt zusätzlich Parameter für die Knochenlänge der Hand anhand eines Eingabebildes zurück. Dadurch wird das kinematische Handmodell automatisch für jeden Nutzer angepasst.

3.8 Zusammenfassung

Für die Handposenerkennung existiert eine Vielfalt an verschiedenen Verfahren. Dabei hat jede Methode ihre Vorteile. Die modellbasierten Ansätze können durch das verwendete Handmodell Einschränkungen auf die zu erkennenden Handposen festlegen und auf diese Weise nur physisch korrekte Handposen zurückgegeben. Allerdings sind modellbasierte Verfahren nicht flexibel, da jede Hand unterschiedliche Knochenlängen besitzt, muss das Verfahren für jede Person neu kalibriert werden. Hybride- und datenbasierten Methoden übertreffen modellbasierte Methoden inzwischen in der Genauigkeit der Erkennung als auch in der Performance. Durch die Hinzunahme von großen öffentlich verfügbaren Datensätzen für Handposen sind datenbasierte Methoden deutlich robuster und genauer geworden. Die Verwendung von neuen Netzwerkarchitekturen wie ResNet und Region Based Networks helfen die Genauigkeit der Erkennung weiter zu erhöhen. Ansätze mit der Verwendung von 3D-CNNs erzielen ebenfalls gute Ergebnisse. In Tabelle 3.3 ist eine Übersicht über die Performance auf dem ICVL Datensatz der hier vorgestellten Verfahren zu finden. Ein Nachteil von datenbasierten Verfahren ist, dass diese im Vergleich zu den modellbasierten Verfahren erst trainiert werden müssen. Dies kann Tage in Anspruch nehmen. Im Gegensatz dazu, muss für modellbasierte Methoden ein passendes Handmodell modelliert werden.

Da datenbasierte Ansätze aktuell die besten Ergebnisse erzielen, werden in dieser Arbeit drei Verfahren solcher Methoden zu einem Vergleich herangezogen: DeepPrior++, Dense-Regression und V2V Posenet. Diese Methoden werden ausgewählt, da sie gute Ergebnisse auf den öffentlich verfügbaren Datensätzen erzielen und der Source Code zur Verfügung steht. Weiterhin unterscheiden sie sich in ihren Ansätzen, DeepPrior++ verwendet ein ResNet in Verbindung mit einem Prior, Dense-Regression verwendet anstelle der direkten Gelenkpunktkoordinaten eine andere Parametrisierung, und V2V Posenet basiert auf einem 3D-CNN. Auf diese Weise können die unterschiedlichen Verfahren und ihre Vorteile und Nachteile miteinander verglichen und untersucht werden. Auf Basis dieser Ergebnisse wird ebenfalls eine eigene Methode entwickelt und mit den eben genannten Verfahren in Bezug auf Genauigkeit, Performance und Zuverlässigkeit evaluiert.

Für alle Verfahren spielt die Handsegmentierung eine essentielle Rolle. Wird eine Region zurückgegeben, in welcher die Hand nicht komplett enthalten ist können nie alle Gelenkpunkte korrekt zurückgegeben werden. Deswegen werden bei fast allen Verfahren Annahmen auf die aufgenommene Szene gemacht, um die Erkennung der Handregion robuster zu machen. Die Verwendung von Tiefenthresholding und der Annahme, dass die Hand sich am nächsten zur Kamera befindet, ist ein Beispiel hierfür. In dieser Arbeit werden deshalb auch die Handsegmentierungsmethoden der Verfahren untersucht. Allgemein lässt sich erkennen, dass die Entwicklung in dem Gebiet der Handerkennung bei Weitem noch nicht abgeschlossen ist. Schnelle Handbewegungen, die vielen verschiedenen Gelenkpunkte und deren Freiheitsgrade stellen bei allen Methoden noch Herausforderungen dar. Deswegen müssen die Verfahren in Zukunft weiterhin verbessert und verfeinert werden.

Tabelle 3.3: Überblick über die erreichte Genauigkeit der vorgestellten Verfahren auf dem ICVL Datensatz. Die erlaubte Abweichung der Gelenkpunkte liegt bei 40 mm.

Quelle	Art	Methodik	Genauigkeit
Tang et al. [TJTK14]	Datenbasiert	Latent Regression Forest	63%
Zhou et al. [ZWZ+16]	Modellbasiert	CNN	84%
Oberweger et al. [OWL15]	Datenbasiert	CNN	90%
Oberweger und Lepetit [OL17]	Datenbasiert	CNN	95%
Guo et al. [GWC+17]	Datenbasiert	Region Ensemble Network	96%
Wan et al. [WPVY17]	Datenbasiert	Hourglass Architektur	97%
Moon et al. [MCL17]	Datenbasiert	3D-CNN	98%

4 Methodik

In diesem Kapitel werden die verwendeten Verfahren zur Handposenerkennung vorgestellt. Dies beinhaltet drei bereits existierende Verfahren und eine eigene selbst entwickelte Methode. Dabei wird auf die verwendete Netzwerkarchitektur, die Besonderheiten der einzelnen Methoden und den verwendeten Datensatz zum Trainieren eingegangen.

4.1 Handposendatensatz

Für das Trainieren der verschieden Verfahren wird der BigHand2.2M Datensatz verwendet. Der Trainingssatz besteht aus 957032 gelabelten Tiefenbilder speziell für die Handposenerkennung. Der Datensatz ist für die ”2017 Hands in the Million Challenge on 3D Hand Pose Estimation” verwendet worden und ist nicht frei öffentlich verfügbar. In dieser Arbeit besteht Zugriff auf den Trainingsdatensatz, in diesem sind für jedes Tiefenbild die 3D-Koordinaten von 21 Gelenkpunkte angegeben. In Abbildung 4.1 sind die verwendeten Gelenkpunkte dargestellt [YYGK17].

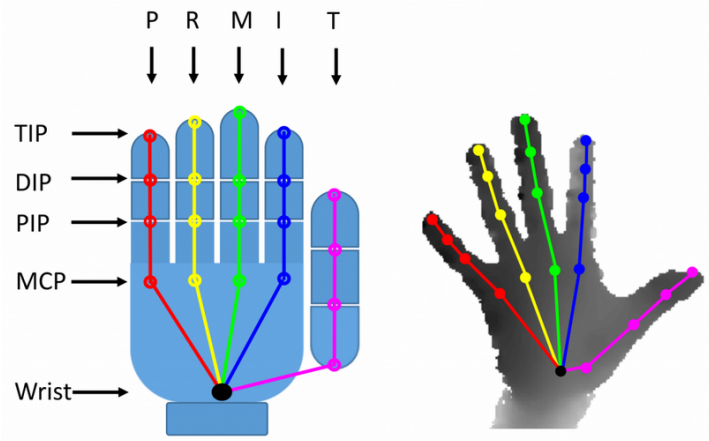


Abbildung 4.1: Verwendete Gelenkpunkte im BigHand2.2M Datensatz. Für jedes Bild existieren die 21 gelabelten 3D-Gelenkpunktkoordinaten [YYGK17].

4.2 Testszenario für die manuelle Montage

Die Handerkennungsalgorithmen werden auf ihre Einsatzfähigkeit im folgenden Szenario untersucht.

In der manuellen Montage besteht ein Bedarf an Assistenzsystemen, welche die Arbeiter bei komplexen Arbeitsdurchläufen unterstützen sollen. Bei Montagevorgängen, welche mehrere hunderte verschiedene Arbeitsschritte beinhalten, besteht die Gefahr, dass ein Arbeiter den Überblick verliert. Beispielsweise können Montageschritte vergessen oder Fehler übersehen werden. Um dies zu verhindern und den gesamten Montagevorgang einfacher gestalten zu können, aber auch in erster Linie um den Arbeiter zu unterstützen, soll ein Assistenzsystem eingesetzt werden. Dieses System dient dazu den Arbeiter bei der Montage zu assistieren und ist mit dem Ziel konzipiert, dass eine interaktive Kommunikation zwischen Arbeiter und System möglich ist. Auf diese Weise kann die Qualität des Produkts erhöht und die Komplexität des Montagevorgangs reduziert werden. Folgende Eigenschaften soll das System mitbringen:

- *Erfassen sämtlicher Arbeitsschritte*
- *Aufzeigen der Arbeitsschrittfolge und direkte Fehlererkennung für den optimalen Prozessablauf*
- *Intuitive Steuerung und Bedienung*
- *Kein Bedarf langer Ausbildung von Mitarbeitern*
- *Multimodale und intuitive Kommunikation zwischen dem Assistenzsystem und dem Arbeiter*

Für die hier untersuchten Handerkennungsalgorithmen wird evaluiert, ob sie in einem solchem Szenario eingesetzt werden könnten. Dabei soll hervorgehen, ob auch in diesen Produktionsumgebungen eine zuverlässige Handerkennung möglich ist. Für die Evaluation wird deshalb ein ähnliches Szenario simuliert. In Abbildung 4.2 ist das Testszenario dargestellt. Es ist wie folgt aufgebaut. In der Mitte einer Werkbank befindet sich ein Sensor der circa in 70cm Höhe montiert ist. Zentral im unteren Bereich befindet sich die Montagefläche, auf welcher der Arbeiter die verschiedenen Bauteile zusammensetzt. Auf der rechten Seite der Werkbank ist das benötigte Werkzeug aufgestellt und auf der linken Seite befinden sich die benötigten Bauteile. Der Bausatz besteht aus einzelnen Metallelementen die zu einem fertigen Produkt zusammengesetzt werden. Hierfür müssen die Bauteile mithilfe von Schrauben und Muttern in der richtigen Reihenfolge zusammengeschraubt werden. Der ganze Montagevorgang wird aufgenommen und die Tiefenbilder und Farbbilder abgespeichert. Da für dieses Szenario keine Wahrheitswerte für die Gelenkkoordinaten vorhanden sind, werden die erkannten Gelenkpunkte in das Farbbild eingezeichnet und visuell ausgewertet, ob es sich um annähernd korrekte Positionen handelt. Dabei werden alle Teilbilder, welche fehlerhaft erkannte Gelenkpunkte beinhalten ausgewertet und mit den als korrekt ausgewerteten Teilbildern gegenübergestellt. Auf diese Weise sollen mögliche Fehlerquellen erkannt und eine allgemeine Schlussfolgerung auf die Genauigkeit der Handerkennungsalgorithmen in diesem Testszenario getroffen werden können.

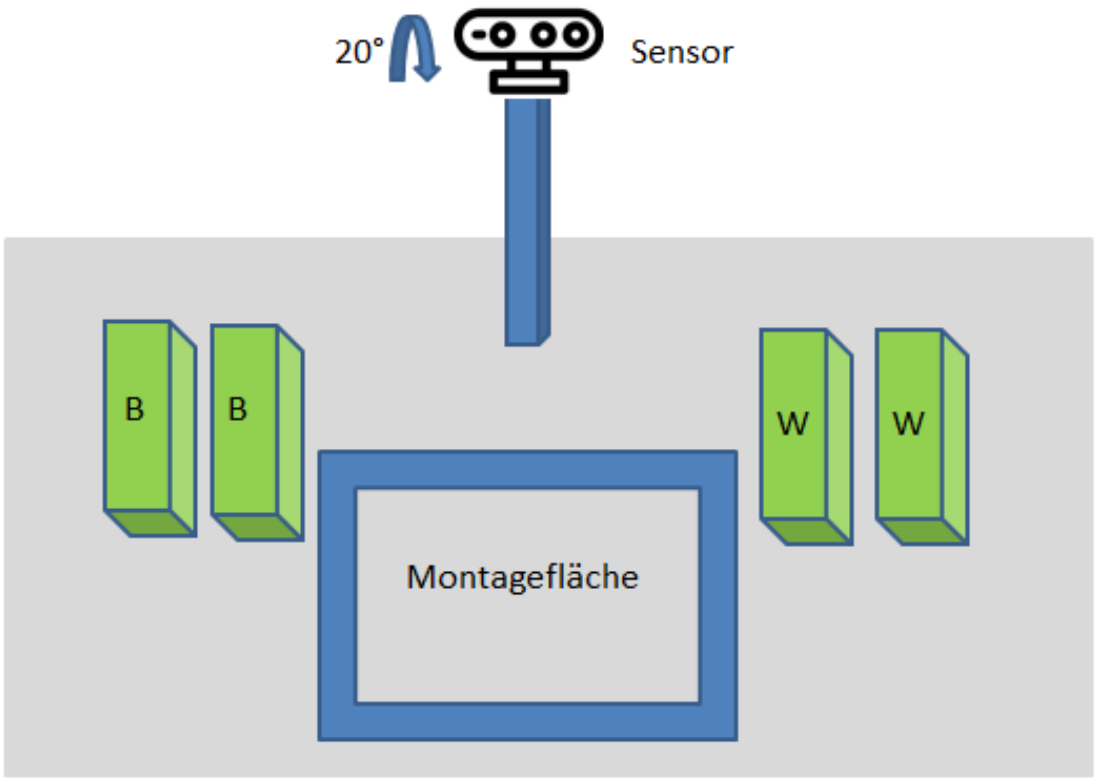


Abbildung 4.2: Aufbau des Testszenarios. Der Sensor ist in der Mitte des Tisches in ca. 80 cm Höhe platziert und um 20° nach unten geneigt. B steht für Bauteile und W für Werkzeuge. Zentral im unteren Bereich befindet sich die Montagefläche.

4.2.1 Sensor

Für das Testszenario wird der Intel RealSense D435 Sensor verwendet. Er nimmt sowohl ein Farbbild, als auch ein Tiefenbild der Szene auf. Das Farbbild wird mit einer Auflösung von 1920x1080 Pixeln und das Tiefenbild mit einer Auflösung von 640x480 Pixeln aufgenommen. Der Sensor gibt die Tiefenwerte in Millimetern zurück und besitzt eine Brennweite von 382.96mm. Die horizontale Sichtweite beträgt 91.2° und die vertikale Sichtweite 65.5°. Dabei besitzt er eine Reichweite von 0.2-10 Metern für die Tiefenerkennung [Int18].

4.3 Handsegmentierung

Alle Methoden, die in diesem Kapitel betrachtet werden, trennen zwischen der Handsegmentierung und der Handposenerkennung auf. Für die Gelenkpunkterkennung wird die segmentierte Handregion des Tiefenbildes als Eingabe benötigt. Beim Training der Netzwerke wird anhand der gelabelten Trainingsdaten die Position der Hand ermittelt und die gewünschte Handregion segmentiert. Hierfür wird das erste Gelenk im Mittelfinger (Metacarpophalangealgelenk) als Zentrum für die Handregion definiert und ein fest definierter Rahmen um die Handregion ausgeschnitten. Dieser Ausschnitt

der Hand aus dem Tiefenbild dient als Eingabe für die Handerkennungsalgorithmen. Auf diese Weise werden beim Trainieren der Netzwerke korrekte Handregionen verwendet. Der Vorgang funktioniert für gelabelte Daten, das heißt die Positionen der Gelenkpunkte in dem Tiefenbild müssen angegeben sein. Für nicht gelabelte Daten wird eine andere Methode zur Extrahierung der Handregion genutzt. Hierfür wird bei allen Verfahren aktuell die Handsegmentierung von DeepPrior++ verwendet [OL17]. Im Folgenden werden die Handsegmentierung von DeepPrior++ und das selbst entwickelte Verfahren vorgestellt.

4.3.1 Handsegmentierung DeepPrior++

Die Handsegmentierung von DeepPrior++ setzt auf ein simples und robustes Verfahren. Es werden Anforderungen an die Handposition gestellt, die für eine zuverlässige Erkennung eingehalten werden müssen. Hierfür wird angenommen, dass die Hand das am nahestehende Objekt zur Kamera ist. Zusätzlich wird nur die Handregion im Bild erkannt, die sich näher am Sensor befindet. Es kann daher nur eine der beiden Hände erkannt werden. Die Handsegmentierung ist wie folgt aufgebaut. Zuerst wird ein Tiefenbereich festgelegt, der stufenweise erhöht wird, bis Pixel gefunden worden, die in diesem Bereich Tiefenwerte besitzen. Daraufhin kann mithilfe der umliegenden Pixel und deren Tiefenwerte das Massenzentrum berechnet werden. Dieses wird als Mittelpunkt der Handregion definiert. Um die Genauigkeit weiter zu steigern, wird zusätzlich ein CNN verwendet, welches die finale Position des Mittelpunkts der Handregion weiter verfeinert. Es bekommt als Eingabe das berechnete Massenzentrum und das Tiefenbild und gibt einen Abweich-Vektor zurück, welcher beschreibt wie weit das Massenzentrum von Mittelpunkt der Hand entfernt ist. Die Architektur dieses CNN ist in Abbildung 4.4 dargestellt. Das Netzwerk besteht aus einem Convolutional-Layer gefolgt von einem MP-Layer und einem Residual Block. Danach folgen FC-Layer und Dropout Layer, welche den Ausgabewert liefern. Mit der Verwendung des CNN kann die Genauigkeit der Handsegmentierung zusätzlich gesteigert werden. Zuvor muss ein ausreichend großer Trainingsdatensatz vorhanden sein um dieses Netzwerk zu trainieren. Beim Einsatz in einem Echtzeitszenario wird das Massenzentrum nicht jedes mal neu berechnet, sondern der bereits bestimmte Mittelpunkt des vorherigen Frames als Input in das CNN verwendet. Dies führt zu einer gleichbleibenden Genauigkeit, verringert die Laufzeit der Methode aber deutlich. In Abbildung 4.3 wird der komplette Prozess der Handsegmentierung von DeepPrior++ dargestellt.

Abbildung 4.3: Prozess der Handsegmentierung von DeepPrior++. Als Eingabe dient ein Tiefenbild einer kompletten Szene, als Ausgabe wird das auf die Handregion ausgeschnittene Tiefenbild zurückgegeben.

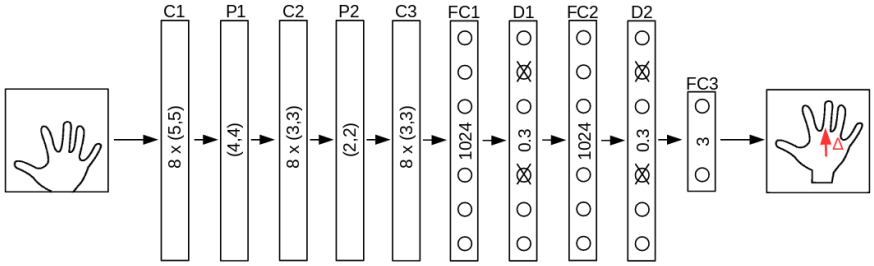


Abbildung 4.4: Netzwerkarchitektur der Gelenkpunktaktualisierung von DeepPrior++. C beschreibt ein Convlutional Layer, FC für ein FC-Layer, D ein Dropout-Layer, R ein Residual Modul und P ein MP-Layer [OL17]. Das Netzwerk gibt einen Abweich-Vektor zurück, der beschreibt in welche Richtung und wie weit der erste Gelenkpunkt im Mittelfinger verschoben werden soll.

4.3.2 Eigene Handsegmentierungmethode

In dem Anwendungsszenario der manuellen Montage ist die Handsegmentierung von DeepPrior++ nicht ausreichend. Die Anforderungen, dass nur eine Hand erkannt werden kann und diese sich am nächsten zur Kamera befinden muss, sind für viele Use-Cases an einer Werkbank nicht erfüllbar. Deshalb ist eine zusätzliche Handsegmentierungsmethode entwickelt worden. Hierzu wird das Erkennungsnetzwerk YoloV3 verwendet [RF18]. Das Netzwerk kann Objekte innerhalb eine RGB Bildes erkennen, in diesem Fall ist es darauf trainiert, für alle Hände in der aufgenommenen Szene, ein umschließendes Rechteck zurückzugeben. Die Architektur von YoloV3 ist in Abbildung 4.6 dargestellt. Es besteht aus 53 Convolutional-Layern und baut zusätzlich auf Residual Blöcke aus der ResNet-Architektur.

Die Methode zur Handsegmentierung ist dabei wie folgt aufgebaut. Zunächst wird ein Sensor benötigt, der sowohl ein RGB-Farbbild, als auch ein Tiefenbild derselben Szene aufnehmen kann. Im Testszenario wird hierfür der Intel RealSense Sensor D435 verwendet. Der erste Schritt besteht aus der Erkennung der Hände im RGB-Farbbild. Das YoloV3 Netzwerk gibt mit dem Farbbild als Input die Koordinaten der umschließenden Rechtecke für die Hände zurück. Im zweiten Schritt wird anhand dieser Rechtecke jeweils das Tiefenbild zugeschnitten. Dieses Tiefenbild dient als Input für die DeepPrior++ Handsegmentierungsmethode. Jetzt wird kein Tiefenthresholding durchgeführt, sondern es kann direkt das Massenzentrum berechnet und das CNN zur Verfeinerung verwendet werden. Mithilfe dieser Kombination der Verfahren kann die Hand sich weiter entfernt als andere Objekte von dem Sensor befinden und muss nicht mehr das am nächstliegendste Objekt zur Kamera sein. Zusätzlich können gleichzeitig mehrere verschiedene Hände im Bild erkannt werden. Der komplette Prozess der Handsegmentierung ist in Abbildung 4.5 dargestellt.

Für das Training des YoloV3 Netzwerks ist der Datensatz aus [GOC17] verwendet worden. Das Trainingsset besteht aus 80420 Bildern und für jedes dieser Bilder sind die Koordinaten des umschließenden Rechtecks der Handregionen angegeben. Die Bilder sind mit einer Auflösung von 640x480 Pixeln aufgenommen worden und für alle Bilder existieren Aufnahmen aus verschiedenen Perspektiven. Hierfür sind mehrere Sensoren zeitgleich verwendet worden.

Abbildung 4.5: Prozess der eigenen Handsegmentierungsmethode. Als Eingabe dient ein Farbbild und Tiefenbild einer kompletten Szene, als Ausgabe wird das auf die Handregion ausgeschnittene Tiefenbild zurückgegeben.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1	
	Softmax			

Abbildung 4.6: Netzwerkarchitektur der YoloV3 Methode. Es werden 53 Convolutional-Layer benutzt, das Netzwerk trägt daher den Namen Darknet-53 [RF18].

4.4 Handerkennungsalgorithmus DeepPrior++

4.4.1 Architektur

DeepPrior++ setzt auf einer ResNet-Architektur. Das hier verwendete Modell basiert auf einem ResNet-50 Netzwerk aus [HZRS16]. Durch die ResNet-Architektur können deutlich tiefere Netzwerke erstellt werden im Vergleich zu herkömmlichen Convolutional Netzwerkarchitekturen wie AlexNet. Dies wird durch die sogenannten ”Skip Connections” erreicht. Eine genauere Beschreibung der Funktionsweise ist im Kapitel ”Grundlagen” zu finden. Da das ResNet für Bildklassifikation entworfen ist, die Rückgabe der Handgelenkpunkte aber ein Regressionsproblem ist, wird in DeepPrior++ die Architektur in den letzten Layer angepasst. Das globale Pooling-Layer wird entfernt und zwei FC-Layer hinzugefügt. Die finale Netzwerkarchitektur besteht aus einem initialen Convolutional-Layer mit 64 Filtern gefolgt von einem 2x2 MP-Layer. Danach folgen vier Residual Module, alle mit einem Stride von 2x2 und jeweils mit 64,128,256,256 Filtern. Nach diesen kommen zwei FC-Layer mit 1024 Parametern und jeweils einem Dropout Layer dazwischen für die Regulation. Den Abschluss bilden ein niedrig dimensioniertes FC-Layer mit 30 Parametern und am Ende dann ein FC-Layer mit den 3D-Gelenkkoordinaten als Output [OL17]. Die gesamte Architektur ist in Abbildung 4.7 dargestellt.

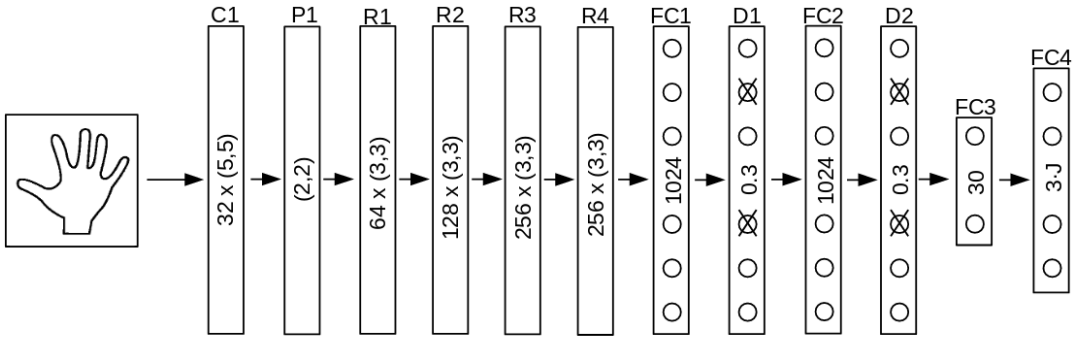


Abbildung 4.7: Netzwerkarchitektur von DeepPrior++ auf Basis eines ResNet. C beschreibt ein Convolutional-Layer, FC für ein FC-Layer, D ein Dropout-Layer, R ein Residual Modul und P ein MP-Layer. [OL17].

4.4.2 Methodik

Das Hauptmerkmal von DeepPrior++ ist die Verwendung eines ”Prior” innerhalb der Netzwerkarchitektur. Dabei werden vor dem Start des Trainings des Netzwerks bereits Informationen über die Handgelenkpunkte verwendet. In diesem Fall wird hierfür im vorletzten Layer ein sogenanntes Bottleneck Layer eingebaut, ein FC-Layer das eine niedrig dimensionierte Repräsentation der Gelenkkoordinaten lernt. Es besteht aus 30 Dimensionen und somit weniger Dimensionen als das finale Output-Layer mit den 3D-Gelenkkoordinaten (bei 21 Gelenkpunkten besitzt das Ouputlayer 3x21= 63 Dimensionen). Für das Mapping von dieser niedrig dimensionierten Repräsentation der Gelenkkoordinaten auf die finalen 3D-Gelenkpunkte wird der "Pose Prior"verwendet. Hierfür wird im Vorhinein PCA (Principal Component Analysis) auf den vorhanden Trainingsdaten der

Hand angewendet und mit den erhaltenen Daten werden die Gewichte vom Bottleneck Layer zum finalen Output-Layer initialisiert. Auf diese Weise soll die Vorhersage von physisch nicht möglichen Handposen eingeschränkt und korrekte Handposen zurückgegeben werden, da durch die niedrige Dimension aufgrund des Bottleneck Layers der Ergebnisraum der physisch falschen Posen deutlich kleiner wird.

Als Input für das Netzwerk dient ein 128x128 Tiefenbild der Handregion. Diese wird durch den Einsatz der bereits vorgestellten Handsegmentierung von DeepPrior++ erzeugt. Die Tiefenwerte werden auf den Bereich -1 bis 1 normalisiert und dann in das Netzwerk gegeben. Für das Training wird Augmentation verwendet, im speziellen werden die Handtiefenbilder rotiert, skaliert und verschoben. Die Augmentierung der Daten wird zur Laufzeit des Trainings durchgeführt, somit sieht das Netzwerk mit jeder Epoche neue Daten. Als Optimierungsfunktion für die Netzwerkparameter wird ADAM mit den Standard Hyperparameter und einer Lernrate von 0.0001 eingesetzt. Das Netzwerk wird für 100 Epochen trainiert [OL17].

4.4.3 Implementierung

Der Quellcode für die Deep-Prior++ Methode steht auf Github öffentlich zur Verfügung [Obe]. Für das Training auf dem BigHand2.2M Datensatz sind zusätzliche Funktionen implementiert worden. Der gesamte Code ist in Python geschrieben und verwendet das Theano-Framework zum Trainieren der Netzwerke. Im Speziellen ist eine Funktion für das Einlesen der 3D-Gelenkkoordinaten und Tiefenbilder des BigHand2.2 Datensatzes hinzugefügt worden. Des Weiteren wird der vorhanden Code für das Training auf dem neuen Datensatz angepasst und erweitert. Hierzu ist der Trainingscode von PCA für den Prior und des ResNet für den neuen Datensatz neu implementiert worden. Für die Evaluation werden Funktionen entwickelt, welche die Abweichung zum Wahrheitswert berechnen und die Gelenkpunkte im Testbild einzeichnen können.

4.5 Handerkennungsalgorithmus Dense-Regression

4.5.1 Architektur

Dense-Regression verwendet eine Netzwerkarchitektur, die auf dem Hourglass Netzwerk basiert [NYD16]. Das Hourglass Netzwerk ist speziell für die Posenerkennung bei Menschen entworfen worden und wird hier auf die Handposenerkennung angewendet. Das Netzwerk ist auf eine Weise aufgebaut, sodass möglichst bei jeder Auflösung Informationen eingefangen werden können. Dafür werden Convolutional-Layer und MP-Layer verwendet, um die Auflösung schrittweise auf ein Minimum zu reduzieren. Sobald die minimale Auflösung erreicht ist, wird mithilfe von Upsampling die Auflösung wieder auf die ursprüngliche Ausgangssituation erhöht. Bei jedem Upsampling Schritt werden die Features der nächstgrößeren Auflösung elementweise addiert. Nachdem die Zielauflösung erreicht ist, folgen zwei 1x1 Convolutions und die finale Ausgabe des Netzwerks wird zurückgeben. Einen Überblick über die Hourglass Architektur ist in Abbildung 4.8 dargestellt. Für eine gegebene Heatmap gibt das Netzwerk die Wahrscheinlichkeit, ob es sich um einen Gelenkspunkt handelt, für jeden einzelnen Pixel zurück. Dense-Regression stapelt zwei solcher Hourglass Module aufeinander. Es werden zwei Hourglass Module verwendet, da die Methode ansonsten nicht mehr in Echtzeit lauffähig ist. Die Netzwerkarchitektur von Dense-Regression ist in Abbildung 4.10

dargestellt. Für die Residual Module wurde ein Kernel von 3x3 verwendet und für das Pooling ein 2x2 MP-Layer eingesetzt. Die Convolutional-Layer sind mit einem 7x7 Kernel und einem Stride von 2x2 ausgestattet [WPVY17].

4.5.2 Methodik

Die Methode von Dense-Regression sticht besonders durch die Parametrisierung der Pose hervor. Sie ist unter der Rücksicht gewählt, sodass sowohl die 2D-, als auch die 3D-Eigenschaften des Tiefenbildes betrachtet werden können. Dies wird erreicht, indem die Handposenparameter durch eine 2D-Heatmap, eine 3D-Heatmap und ein dreidimensionales direktionales Vektorfeld dargestellt werden. Eine weitere Besonderheit ist, dass die 3D-Gelenkpunkte nicht direkt durch Regression abgeleitet werden, sondern ein Offsetvektor zwischen den Tiefenwerten und den Gelenkpunkten der Hand bestimmt wird. Dadurch ist die Vorhersage des Netzwerks translationsinvariant, das heißt bei einer Translation (Verschiebung) der Handposenparameter sich die Ausgabe des Netzwerks nicht ändert. Zusätzlich soll sich diese Parametrisierung besser für verschiedene Kombinationen an Fingerposen verallgemeinern. Eine direkte Regression des dreidimensionalen Offsetvektors ist nicht ideal. Regression für Punkte die weit entfernt von den Handgelenken liegen resultieren in Offsetvektoren mit großen Normen, welche beim Training des CNNs den abgeleiteten Fehler dominieren würden. Deswegen wird der Offsetvektor in zwei Komponenten zerlegt, eine 3D-Heatmap und einen direktionalen Einheitsvektor.

Durch die 2D-Heatmap werden die lokalen geometrischen Oberflächenmuster eingefangen. Von der 3D-Perspektive wird das Tiefenbild als 3D-Punktwolke betrachtet und von dieser der Einheitsvektor und eine 3D-Heatmap abgeleitet. Der Einheitsvektor wird durch Regression bestimmt und die Gelenkpunkte auf den Heatmaps durch Erkennung abgeleitet. Alle Ausgaben des Netzwerks werden mithilfe des Mean-Shift-Algorithmus in eine globale Vorhersage überführt mit Übereinstimmung zwischen den 2D- und 3D-Ausgaben. Der finale Output ist dann ein 3D-Offsetvektor, welcher für jeden Pixel angibt wie weit dieser von einem Gelenkpunkt entfernt ist. Als Input für das Netzwerk dient ein Tiefenbild mit einer Auflösung von 128x128. Diese kommt durch ein Convolutional-Layer und wird dann mit einer Dimension von 32x32 in das Hourglass Modul gegeben. Beim Training wird Augmentierung eingesetzt, die Tiefenbilder werden zufällig rotiert und skaliert. Als Optimierungsfunktion kommt ADAM zum Einsatz mit eine Lernrate von 0.001.

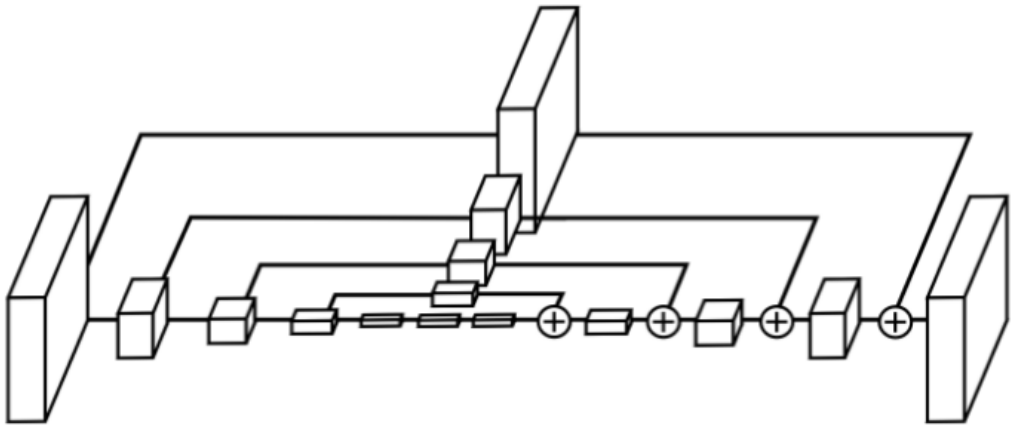


Abbildung 4.8: Architektur eines Hourglass Netzwerks. Jeder Block im Netzwerk besteht aus einem ”Residuale Module”, welches in Abbildung 4.9 dargestellt ist. Die Anzahl der Features ist konsistent über das gesamte Netzwerk. [NYD16].

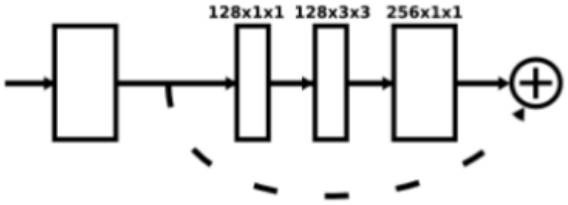


Abbildung 4.9: Aufbau eines einzelnen Hourglass Moduls. Das Design ist an ein ”Residual Module” angelehnt und verwendet Skip Connections [NYD16].

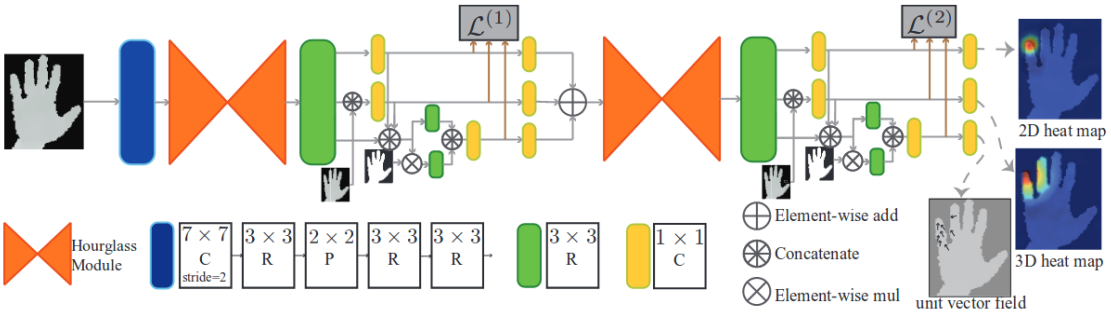


Abbildung 4.10: Netzwerkarchitektur des Dense-Regression Netzwerks. Die Abkürzungen C,P,R stehen für Convolution Layer, Pooling-Layer und Residuale Module. Das Netzwerk bestimmt eine 2D-Heatmap, eine 3D-Heatmap und einen Einheitsvektorfeld für jeden Gelenkpunkt. In dieser Abbildung wird die Berechnung des Gelenkpunkts im kleinen Finger dargestellt [WPVY17].

4.5.3 Implementierung

Der Quellcode für die Dense-3D-Regression Methode steht auf Github öffentlich zur Verfügung [Mel]. Für das Training auf dem BigHand2.2M Datensatz sind Scripts ergänzt worden. Der gesamte Code ist in Python geschrieben und verwendet das Tensorflow-Framework zum Trainieren des Netzwerks. Für das Training selbst sind die Trainingsbilder und die Gelenkpunkte in das TF-Record Format umgewandelt worden. Dies erlaubt eine schnellere Performance beim Einlesen der Daten im Trainingsvorgang. Der Code für das Training ist für den neuen Datensatz erweitert und angepasst worden. Für die Evaluation werden Funktionen implementiert, welche die Differenz zu den Wahrheitswerten berechnen und die Gelenkpunkt in das Testbild einzeichnen können.

4.6 Handerkennungsalgorithmus V2V-PoseNet

4.6.1 Architektur

Die Architektur von V2V-PoseNet baut ebenfalls auf dem Hourglass Netzwerk auf. Sie wurde leicht angepasst und ist für 3D-Input aufgebaut. Das heißt die z-Achse wird als zusätzliche räumliche Achse betrachtet, sodass die Kernel-Größe durch Weite x Höhe x Tiefe definiert ist. Zuerst folgt ein 3D-Convolutional-Layer und ein volumetrischer Downsampling Block der den Input auf die gewünschte Größe downsampled. Daraufhin sollen drei volumetrische Residual Blöcke die sinnvollen Eigenschaften extrahieren. Dies dient als Input für das Hourglass Modul. Hier wird der Downsampling-Part als Encoder und der Upsampling-Part als Decoder definiert. In Abbildung 4.12 sind diese detaillierter dargestellt. Im Encoder reduziert der volumetrische Downsampling Block die Größe der Featuremap, während die Residual Blöcke die Anzahl der Channel der Featuremaps erhöhen. Auf der anderen Seite im Decoder wird das Gegenteil durchgeführt. Die Featurmaps werden upgesamplet und die Anzahl der Channel reduziert. Auch hier wird, wie beim Hourglass Netzwerk, bei jeder Skalierung elementweise Addition mit dem vorherigen Block durchgeführt. Nach dem Decoder wird mithilfe von einem Residual Block und zwei 1x1x1 3D-Convolutional-Layern die Wahrscheinlichkeit, welche angibt um welchen Gelenkpunkt es sich handelt, für jeden Voxel zurückgegeben. Die komplette Architektur ist in Abbildung 4.11 bildlich dargestellt.

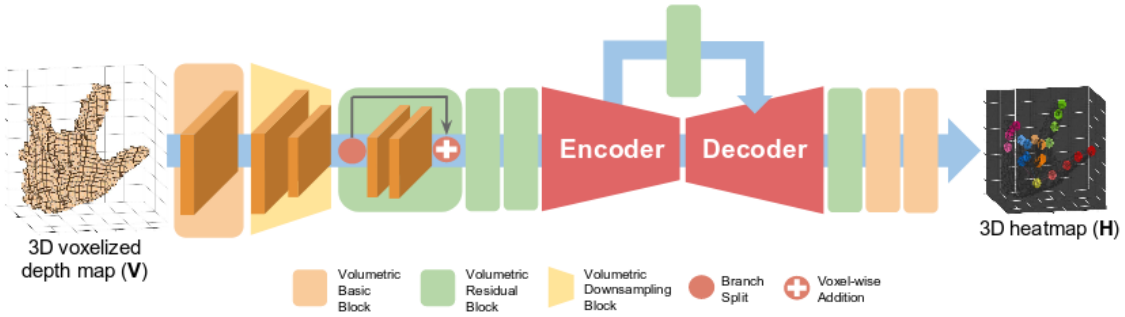


Abbildung 4.11: Netzwerkarchitektur des 3D-CNN mit Voxeln. Die Basisblöcke stehen für ein 3D-Convolutional-Layer. Für jeden Voxel wird ein Wahrscheinlichkeitswert berechnet der angibt wie wahrscheinlich er ein Gelenkpunkt ist und am Ende wird darauf basieren eine 3D-Heatmap zurückgegeben. Jede Farbe in der Heatmap steht für Gelenkpunkte in demselben Finger [MCL17].

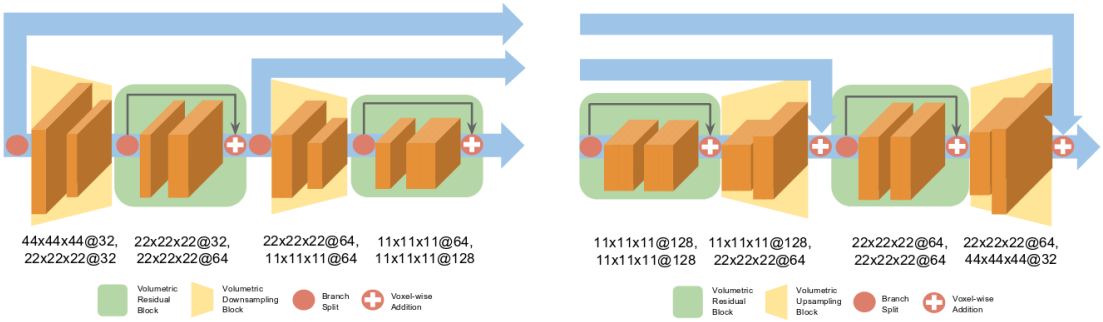


Abbildung 4.12: Decoder und Encoder des V2V Pose Netzwerks. Auf der linken Seite ist der Encoder und auf der rechten Seite der Decoder abgebildet. Die Zahlen unterhalb stellen die räumliche Größe gefolgt von der Anzahl der Featuremaps [MCL17].

4.6.2 Methodik

Das V2V Pose Netzwerk sticht hervor durch die Nutzung einer 3D-CNN-Architektur und der Verwendung einer Voxel-Repräsentation der Daten als Input in das Netzwerk. Ein Überblick dazu ist in Abbildung 4.13 dargestellt. Zuerst wird das Tiefenbild in eine dreidimensionale volumetrische Form konvertiert, indem die Punkte mithilfe der Tiefenwerte in den 3D-Raum projiziert werden. Als Output wird bei allen Gelenkpunkten für jeden Voxel die Wahrscheinlichkeit, ob es sich um einen Gelenkpunkt handelt, ausgegeben. Die Voxel mit der höchsten Wahrscheinlichkeit werden identifiziert und als 3D-Koordinaten zurückgegeben. Als Input in das Netzwerk wird eine Voxel Repräsentation des Tiefenbildes mit einer Größe von 88x88x88 definiert. Für die Optimierungsfunktion wird RMSProp mit einer Lernrate von 0,00025 verwendet. Beim Training werden die Daten augmentiert in Form von Rotation, Skalierung und Translation.

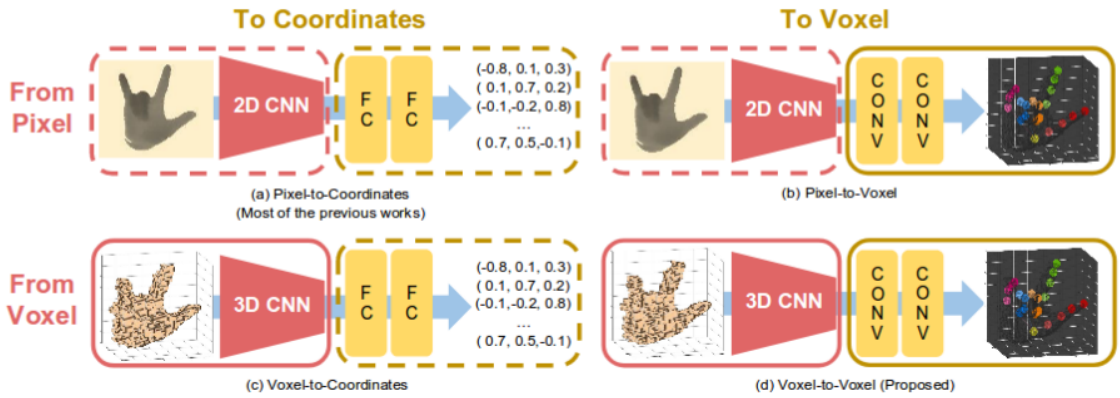


Abbildung 4.13: Überblick über Kombination von Eingabe und Ausgabe der Handerkennungsalgorithmen anhand eines einzelnen Tiefenbildes. Die meisten bisherigen Verfahren setzten auf Methode (a). Das V2V Pose Netzwerk setzt bei Eingabe und Ausgabe beides mal auf Voxel eine 3D-Repräsentation [MCL17].

4.6.3 Implementierung

Der Quellcode für die V2V-PoseNet Meth*ode steht auf Github öffentlich zur Verfügung [mks]. Für das Training auf dem BigHand2.2M Datensatz sind Scripts ergänzt worden. Der gesamte Code ist in Lua geschrieben und verwendet das Torch-Framework zum Trainieren des Netzwerks. Für das Einlesen der Tiefenbilder sind diese in ein binäres Format umgewandelt worden um den Vorgang zu beschleunigen. Der Code für das Training auf dem neuen Datensatz ist angepasst und erweitert worden. Die V2V-PoseNet Methode ist bereits auf dem BigHand2.2M Datensatz evaluiert worden, der hierfür verwendete Code ist nicht veröffentlicht und deshalb ergänzt worden. Der Code für das Training ist erweitert und angepasst und für die Evaluation die entsprechenden Funktionen zum Anzeigen der Gelenkkoordinaten und Berechnung der Differenz zu den Wahrheitswerten hinzugefügt worden.

4.7 Handerkennungsalgorithmus des selbst entwickelten Verfahrens

4.7.1 Architektur

Die Architektur baut auf einem ResNeXt Netzwerk auf. Die ResNeXt-Architektur ist eine Weiterentwicklung der Resnet-Methode. In Abbildung 4.14 ein Vergleich zwischen einem ResNet-Block und einem ResNext-Block dargestellt. Ein ResNext Block besitzt im Vergleich mehrere Pfade die am Ende wieder zusammengefügt werden. Die Anzahl der Pfade wird durch den Parameter Kardinalität beschrieben. Die Verwendung dieser zusätzlichen Pfade führt zu einer höheren Genauigkeit im Bereich der Bildklassifizierung [XGD+17].

Die ResNext-Architektur wurde gewählt inspiriert von den Methoden, die das Problem in mehrere Teilbereiche aufteilen und basierend auf allen Teilergebnissen das Endergebnis ableiten. Im Kapitel ”Stand der Technik” sind diese Methoden näher erläutert, wie zum Beispiel das Region

Ensemble Network oder der MuliView Ansatz aus [GLYT16; GWC+17]. Die Hand lässt sich ebenfalls leicht in mehrerer Teilregionen aufteilen, jeder Finger besitzt eine unterschiedliche Länge und lässt sich auf den ersten Blick von den anderen Fingern unterscheiden. Aus diesem Grund sollte eine Netzwerkarchitektur wie ResNeXt gut geeignet sein für das Problem der Handgelenkspunkterkennung und ist für die eigene Methode in dieser Arbeit ausgewählt worden.

In Abbildung 4.15 ist ein Überblick über die gesamte Netzwerkarchitektur dargestellt. Jeder ResNextBlock hat eine Kardinalität von 16 und besteht aus Convolutional-Layer mit Skip-Connections. Auf den Output des letzten ResNext-Block folgt ein Dropout-Layer und ein Flatten-Layer, welches alle Parameter in eine eindimensionale Vektorform bringt. Diese werden durch zwei FC-Layer mit jeweils einem Dropout-Layer dazwischen geschickt. In dem letzten Layer folgt die Ausgabe der 63 Gelenkpunktkoordinaten.

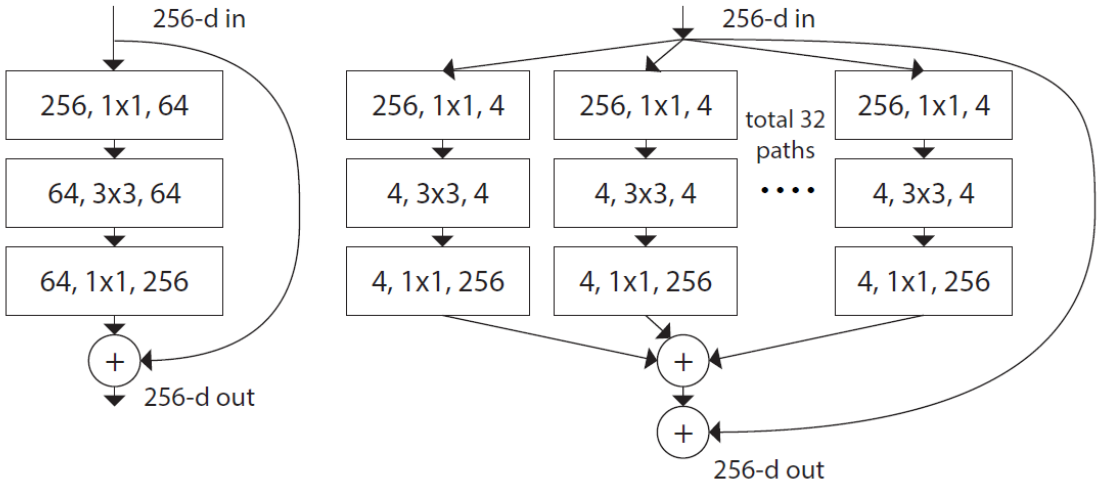


Abbildung 4.14: Vergleich zwischem einem ResNet-Block und einem ResNeXt-Block. Der ResNext-Block teil sich in mehrere Pfade auf die am Ende wieder zusammengefügt werden . Dies wird durch den Parameter Kardinalität ausgedrückt, welcher beschreibt wie viele Pfade sich in einem Block befinden. In diesem Beispiel besitzt der ResNeXt-Block eine Kardinalität von 32 [XGD+17].

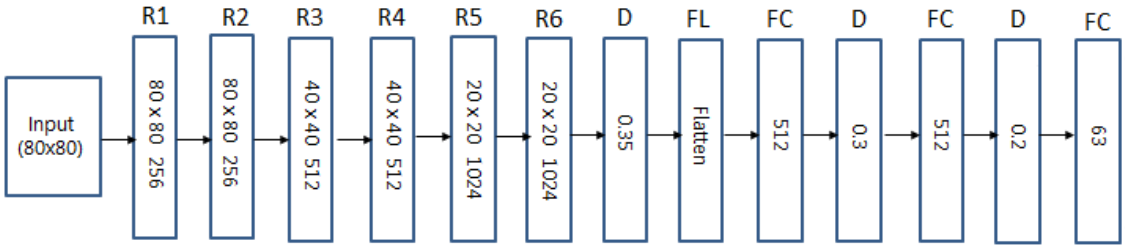


Abbildung 4.15: Netzwerkarchitektur der eigenen Methode. R steht für einen ResNext-Block, FC für FC-Layer, D für Dropout-Layer und FL für Flatten-Layer.

4.7.2 Methodik

Als Eingabe in das Netzwerk kommt ein auf 80x80 skaliertes Tiefenbild der extrahierten Handregion. Für das Training wird die Handregion anhand der gegebenen Wahrheitswerte für die Gelenkposition der Hand extrahiert. Dieses Tiefenbild wird durch das ResNeXt-Netzwerk geleitet, wo direkt die 3D-Gelenkkoordinaten ableitet werden. Hierfür sind am Ende des Netzwerks drei FC-Layer und drei Dropout-Layer zur Regulation hinzugefügt worden. Die Ausgabe ist ein Vektor mit den 63 Gelenkpunktkoordinaten (21 Gelenkpunkte *3 (x,y,z)) der Hand. Für das Training selbst wird als Optimierungsfunktion Adam mit einer Lernrate von 0.001 verwendet.

4.7.3 Implementierung

Der gesamte Code is in Python geschrieben. Für das Training des Verfahrens wird Tensorflow und die High-Level-Api Keras verwendet. Der Code teilt sich in folgende Teilfunktionen auf:

1. Extrahieren der Handregion aus den Tiefenbildern des BigHand2.2M Datensatz
2. Erstellung des Netzwerkmodells basierend auf dem ResNeXt-Netzwerk
3. Trainieren des Netzwerks auf dem BigHand2.2M Datensatz
4. Evaluation und Anzeigen der 3D-Gelenkkoordinaten

5 Evaluierung

In diesem Kapitel werden die Ergebnisse der verwendeten Verfahren vorgestellt. Hierfür werden die Methoden sowohl auf dem Testdatensatz, als auch auf Basis des bereits vorgestellten Testszenarios ausgewertet.

5.1 Datensatz für das Training und Testen

Für die Evaluation ist der BigHand2.2M Datensatz wie folgt aufgeteilt worden.

- 757032 Bilder für das Trainingsset
- 80000 Bilder für das Validierungsset
- 120000 Bilder für das Testset

Basierend auf der Auswertung der Ergebnisse auf dem Validierungsset ist für alle Verfahren beim Training die Epoche des Trainingsvorgangs ausgewählt worden, welche die höchste Genauigkeit auf dem Validierungsset vorweisen konnte. Danach sind alle Verfahren anhand des Testsets ausgewertet worden. In den folgenden Abschnitten werden die Ergebnisse der einzelnen Methoden vorgestellt. Für die Auswertung sind Wahrheitswerte des Datensatzes mit den erzielten Vorhersagen der Methoden verglichen und die Differenz der Ergebnisse untersucht worden.

5.2 Auswertung der Methoden auf dem Testdatensatz

5.2.1 Methode DeepPrior++

Die Ergebnisse der Methode DeepPrior++ sind in Abbildung 5.1 dargestellt. Für jeden Gelenkpunkt ist die mittlere Abweichung vom Wahrheitswert in Millimetern angegeben. Dabei ist die mittlere Abweichung auf jeder Ebene (x,y,z) ausgewertet. Für alle Gelenkpunkte zusammen ergibt sich folgende mittlere Abweichung:

- Mittlere Abweichung auf x-Achse: 4,28 mm
- Mittlere Abweichung auf der y-Achse: 18.8 mm
- Mittlere Abweichung auf der z-Achse: 4,67 mm

Es fällt auf, dass die mittlere Abweichung von 18.8 mm auf der y-Ebene um ein Vielfaches größer ist als auf der x- und z-Ebene. Bei näherer Betrachtung der y-Werte stellte sich heraus, dass bei einem kleinem Anteil der Testbildern der y-Wert extrem abwich und deshalb die Ergebnisse auf dieser Ebene stark abweichen. Da der verwendete Code auf Github [Obe] nicht für diesen Datensatz optimiert ist, sollte mit mehr Feintuning der Parameter noch bessere Ergebnisse erzielt werden können. Die mittlere Abweichung der x- und z-Ebene liegt bei durchschnittlich etwas mehr als 4 mm. Die Genauigkeit auf diesen Ebenen ist als durchweg gut einzustufen. Bei der Betrachtung der Abweichung jedes einzelnen Gelenkpunkts fällt auf, dass die mittlere Abweichung bei den Gelenkpunkten in den Fingerspitzen am höchsten ist (bis zu 7 mm), während die mittlere Abweichung bei den Gelenkpunkten, welche sich im Zentrum der Hand befinden, am geringsten ist. Beim MMCP (siehe Abbildung 4.1) beträgt die mittlere Abweichung etwa 1 mm auf der x- und z-Ebene. Ein Grund für dieses Verhalten, könnte das Tiefenbild selbst sein. An den Randregionen der Hand, ist das Tiefenbild deutlich ungenauer als im Zentrum der Handregion. In Abbildung 5.2 ist ein Beispiel für diese Eigenschaft der Tiefenbilder dargestellt. Auf Grund der Ungenauigkeit in den Randregionen im Tiefenbild, ist es schwerer für das Netzwerk die korrekten Merkmale im Tiefenbild abzuleiten, was sich wiederum in der Genauigkeit an diesen Gelenkpunkten widerspiegelt.

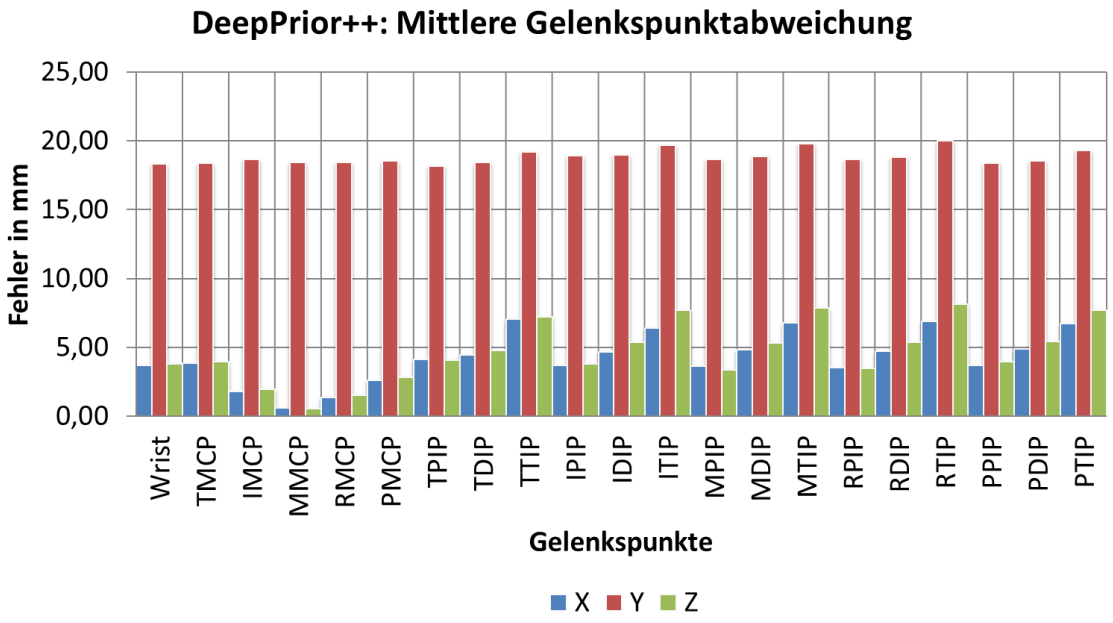


Abbildung 5.1: Auswertungsergebnisse von DeepPrior++ auf dem Testdatensatz. Für jeden Gelenkpunkt ist die mittlere Abweichung in Millimetern auf der x-, y- und z-Ebene dargestellt.

Abbildung 5.2: Ungenauigkeiten an den Randregionen der Hand im Tiefenbild. Die Kontur an den Randregionen ist deutlich ungleichmäßiger als in der Mitte der Hand. Die roten Kreise symbolisieren Beispielstellen für solche Regionen. Das Tiefenbild stammt aus dem NYU-Handdatensatz [YYS+17].

5.2.2 Methode Dense-Regression

Die Ergebnisse der Methode Dense-Regression sind in Abbildung 5.3 dargestellt. Für jeden Gelenkpunkt ist die mittlere Abweichung vom Wahrheitswert in Millimetern angegeben. Dabei ist die mittlere Abweichung auf jeder Ebene (x,y,z) ausgewertet. Für alle Gelenkpunkte zusammen ergibt sich folgende mittlere Abweichung:

- Mittlere Abweichung auf x-Achse: 3,75 mm
- Mittlere Abweichung auf der y-Achse: 3,97 mm
- Mittlere Abweichung auf der z-Achse: 8,30 mm

Die höchste mittlere Abweichung zu den Wahrheitswerten liegt auf der z-Ebene mit 8.3 mm. Danach folgt die y-Ebene mit 3,97mm und die x-Ebene mit 3,75mm. Auch hier ist die mittlere Abweichung an den Gelenkpunkten der Randregionen etwas höher, wobei der Unterschied weniger stark ausfällt als in dem DeepPrior++ Verfahren. Bei der einzelnen Betrachtung jedes Gelenkpunkts fällt auf, das vor allem die Genauigkeit bei den Gelenkpunkten im kleinen Finger und im Daumen geringer ist als bei den anderen Gelenkpunkten. Die Gelenkpunkte im Daumen haben die höchste mittlere Abweichung aller Gelenkpunkte. Die beste Genauigkeit erzielt, wie bei DeepPrior, der MMCP Gelenkpunkt (x: 2.8mm, y: 3,1mm, z: 5,4mm). Auch dieses Verhalten lässt sich wahrscheinlich auf

die Ungenauigkeiten bei den Randregionen im Tiefenbild zurückführen. Allgemein betrachtet ist die mittlere Abweichung bei allen Gelenkpunkten deutlich konstanter und weicht nicht weniger stark voneinander ab, wie bei dem DeepPrior++ Verfahren. Dies trifft vor allem auf die x- und y-Ebene zu.

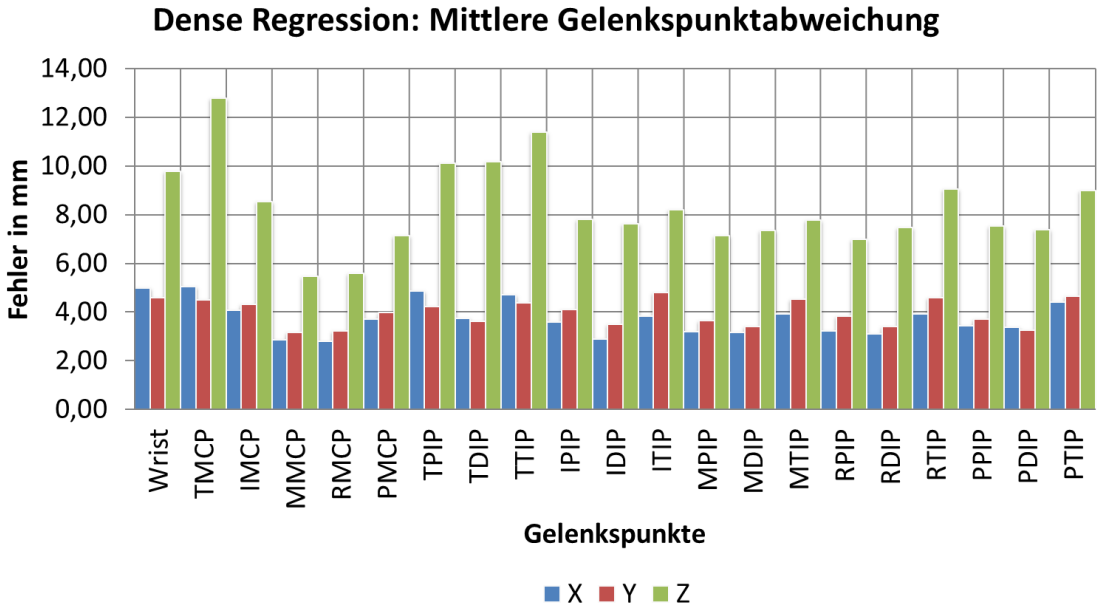


Abbildung 5.3: Auswertungsergebnisse von Dense-Regression auf dem Testdatensatz. Für jeden Gelenkpunkt ist die mittlere Abweichung in Millimetern auf der x-, y- und z-Ebene dargestellt.

5.2.3 Methode V2V-PoseNet

Die Ergebnisse der Methode V2V-Pose sind in Abbildung 5.4 dargestellt. Für jeden Gelenkpunkt ist die mittlere Abweichung vom Wahrheitswert in Millimetern angegeben. Dabei ist die Abweichung auf jeder Ebene (x,y,z) ausgewertet. Für alle Gelenkpunkte zusammen ergibt sich folgende mittlere Abweichung:

- Mittlere Abweichung auf x-Achse: 27,08 mm
- Mittlere Abweichung auf der y-Achse: 36,75 mm
- Mittlere Abweichung auf der z-Achse: 32,01 mm

Die erzielten Ergebnisse fallen negativ aus. Dies ist auf einen Fehler beim Training, beim Einladen der Daten oder im Code von Github [mks] zurückzuführen. Die hier erzielten Ergebnisse sind daher nicht repräsentativ für das Verfahren. V2V-PoseNet ist bereits auf dem BigHand2.2M Datensatz ausgewertet worden und erzielte die höchste Genauigkeit aller Verfahren [YGS+18]. Der dafür verwendete Code ist nicht veröffentlicht, für den weiteren Vergleich werden deshalb die Werte aus [YGS+18] verwendet. Bei Gegenüberstellung der Ergebnisse ist zu beachten, dass sowohl

der Trainingsdatensatz, als auch der Testdatensatz sich von den in dieser Arbeit verwendeten Datensätzen unterscheiden, sie sind allerdings alle aus dem BigHand2.2M Datensatz. Der in [YGS+18] verwendete Trainingsdatensatz besteht aus 957000 Bildern und der Testdatensatz aus 249000 Bildern. Hier erzielte die Methode eine mittlere Abweichung von 7,1 mm bezogen auf alle Gelenkpunkte. In der Auswertung ist die gesamte mittlere Abweichung betrachtetet worden. Des Weiteren wird in der Evaluation zwischen verdeckten Gelenkpunkten und nicht verdeckten Gelenkpunkten unterschieden. V2V-PoseNet erreicht bei Testdaten mit verdeckten Gelenkpunkten eine mittlere Abweichung von 8 mm, bei Testdaten mit nur sichtbaren Gelenkpunkten eine Abweichung von 6.1 mm. Die gesamte mittlere Abweichung setzt sich aus diesen beiden Werten zusammen.

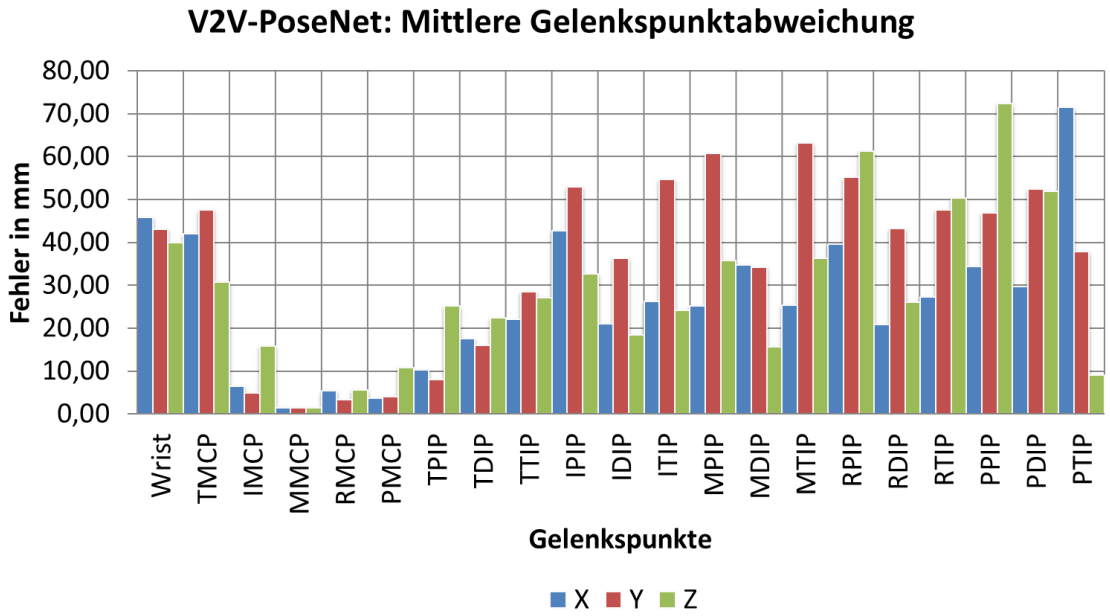


Abbildung 5.4: Auswertungsergebnisse von V2V-PoseNet auf dem Testdatensatz. Für jeden Gelenkpunkt ist die mittlere Abweichung in Millimetern auf der x-, y- und z-Ebene dargestellt.

5.2.4 Eigene Methode

Die erzielten Ergebnisse sind in Abbildung 5.5 dargestellt. Für jeden Gelenkpunkt ist die mittlere Abweichung vom Wahrheitswert in Millimetern angegeben. Dabei ist die Abweichung auf jeder Ebene (x,y,z) ausgewertet. Für alle Gelenkpunkte zusammen ergibt sich folgende mittlere Abweichung:

- Mittlere Abweichung auf x-Achse: 2,88 mm
- Mittlere Abweichung auf der y-Achse: 3,75 mm
- Mittlere Abweichung auf der z-Achse: 14,09 mm

Das Verfahren erzielte die höchste Genauigkeit auf der x- und y-Achse. Die Abweichung auf der z-Achse ist hoch im Vergleich zu den anderen Methoden. Auch hier ist der Trend erkennbar, dass vor allem bei den Gelenkpunkten in den Fingerspitzen die Ungenauigkeit der Vorhersage steigt. Dies betrifft vor allem die z-Ebene, was sich ebenfalls auf die Ungenauigkeit des Tiefenbilds an den Randregionen zurückführen lässt. Die mittlere Abweichung auf der x- und y-Ebene ist ebenfalls höher in den Randregionen der Hand, weicht aber weniger stark ab. Im Vergleich zu den anderen Methoden, konnte dieses Verfahren die z-Ebene, das heißt die Entfernung des Gelenkpunkts zur Kamera, deutlich ungenauer bestimmen. Die mittlere Abweichung auf der x- und y-Ebene ist aber mit 2,88 mm bzw. 3,75 mm am geringsten im Vergleich zu den Methoden DeepPrior++ und Dense-Regression.

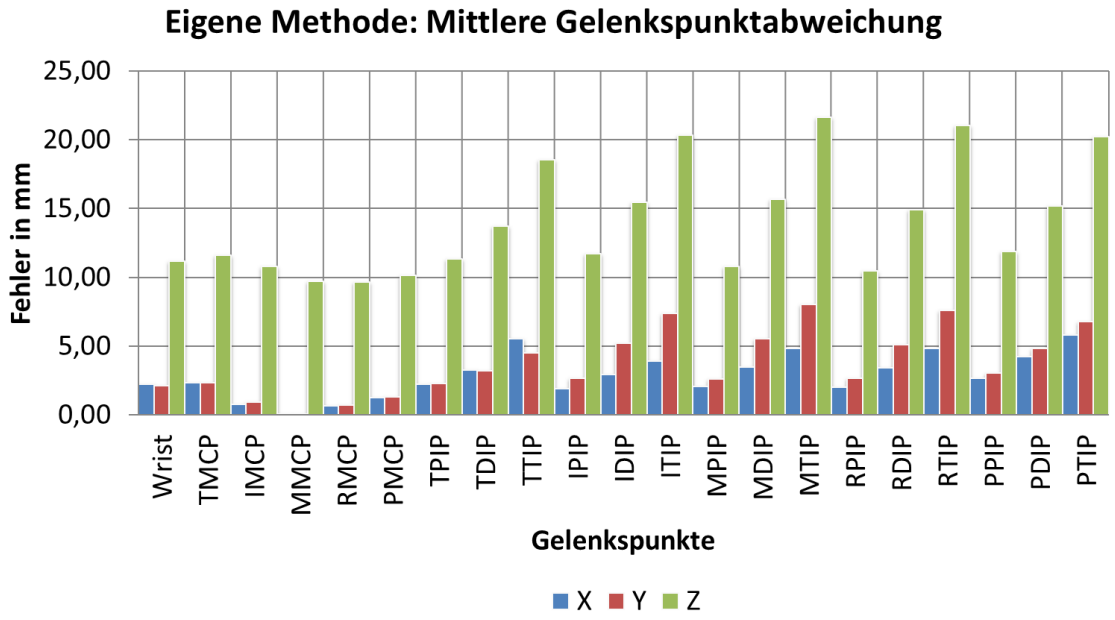


Abbildung 5.5: Auswertungsergebnisse der eigenen Methode auf dem Testdatensatz. Für jeden Gelenkpunkt ist die mittlere Abweichung in Millimetern auf der x-, y- und z-Ebene dargestellt.

5.2.5 Zusammenfassung der Ergebnisse auf dem Testdatensatz

Das V2V-PoseNet Verfahren erzielt mit einer gesamten mittleren Abweichung von nur 7.1 mm die genauesten Ergebnisse. Da das Verfahren in dieser Arbeit aber nicht korrekt nach-implementiert werden konnte, stammen diese Ergebnisse aus einer anderen Auswertung. Zwischen den Methoden Dense-Regression, DeepPrior++ und der selbst entwickelten Methoden erzielte das Dense-Regression Verfahren mit Einbezug aller Ebenen (x:3.75mm, y:3.97mm, z:8.3mm) die besten Ergebnisse. Das selbst entwickelte Verfahren war zwischen den dreien das genauste auf der x- und y-Ebene (x:2.88mm, y:3.75mm, z:14.09mm). Auf der z-Ebene hingegen war es das ungenauste der verglichenen Methoden. DeepPrior++ konnte die besten Ergebnisse auf der z-Ebenen erzielen (x:4.28mm, y:18.8mm, z:4,67mm) war auf den x- und y-Ebene aber das ungenauste Verfahren. Bei allen

Verfahren waren die Ergebnisse in den Randregionen der Hand am ungenauesten. Dazu zählen die Fingerspitzen, der kleine Finger und der Daumen. Die genausten Ergebnisse sind in den Gelenkpunkten im Zentrum der Hand erzielt worden.

5.3 Auswertung Testszenario

5.3.1 Ergebnisse des Testszenarios

Bei der Auswertung wird im Folgenden die Handsegmentierung und die Handgelenkpunkterkennung getrennt betrachtet. Wenn die Handsegmentierung nicht das korrekt Ergebnis zurückgibt kann die Handgelenkpunkterkennung folglich kein korrektes Ergebnis liefern.

5.3.2 Auswertung Handsegmentierung

Die Handsegmentierung im Testszenario funktionierte grundsätzlich zuverlässig. Es sind größtenteils beide Hände erkannt worden. Ungenauigkeiten gab es bei den zurückgegebenen Handregionen. Die Regionen waren teilweise zu klein oder es sind mehr als zwei Handregionen erkannt worden. In Abbildung 5.6 ist ein Beispiel für solche Fehlerfälle dargestellt. Um die richtigen Handregionen auszuwählen, ist deshalb die Regel festgelegt worden, die zwei größten erkannten Regionen als Ergebnis auszuwählen.

Die meisten Fehler bei der Segmentierung im Farbbild traten auf, wenn die Hände sich nah beieinander befanden. In Abbildung 5.7 sind solche Fehler dargestellt. Befanden sich die Hände zu nah beieinander, ist häufig nur eine Handregion erkannt worden, oder eine große Region die beide Hände umschlossen hat (siehe Abbildung 5.7). Des Weiteren sind in den zurückgegebenen Regionen kleine Teile der anderen Hand enthalten gewesen, was wiederum zu Fehlern in der Handgelenkpunkterkennung führte. In diesen Fällen ist die Handsegmentierungsmethode nicht zuverlässig genug gewesen.

Allgemein lässt sich feststellen, dass die Handsegmentierung zuverlässig funktioniert, sofern sich die Hände nicht zu nah beieinander befinden. Die bereits erwähnten Fehlerfälle halten sich in Grenzen und treten nicht bei jedem einzelnen Bild aus dem Testszenario auf, es gibt auch positive Ergebnisse wenn beide Hände nah beieinander sind. In Abbildung 5.8 sind Beispiele für korrekt erkannte Handregionen abgebildet.

5.3.3 Auswertung Handgelenkpunkterkennung

Die Handgelenkpunkterkennung im Testszenario ist in den meisten Fällen zu ungenau gewesen. Keine der Verfahren konnte zuverlässig die Position der Handgelenkpunkte bestimmen. Wenn die Hände sich nahe zu Tischplatte befanden, waren die Resultate der Methoden am schlechtesten. In Abbildung 5.9 sind Beispiele hierfür dargestellt. Allgemein war die Bestimmung der 3D-Gelenkpunktkoordinaten zu ungenau und erzielt deutlich schlechtere Ergebnisse als auf dem Testdatensatz. Da für die Bilder im Testszenario keine Wahrheitswerte vorhanden sind, ist die

Auswertung visuell erfolgt, indem die erkannten Gelenkpunkte in das Testbild eingezeichnet worden sind. Trotzdem ist klar zu erkennen gewesen, dass die bestimmten Gelenkpunktkoordinaten in vielen Fällen oft deutlich vom Wahrheitswert abgewichen sind (siehe Abbildung 5.9)

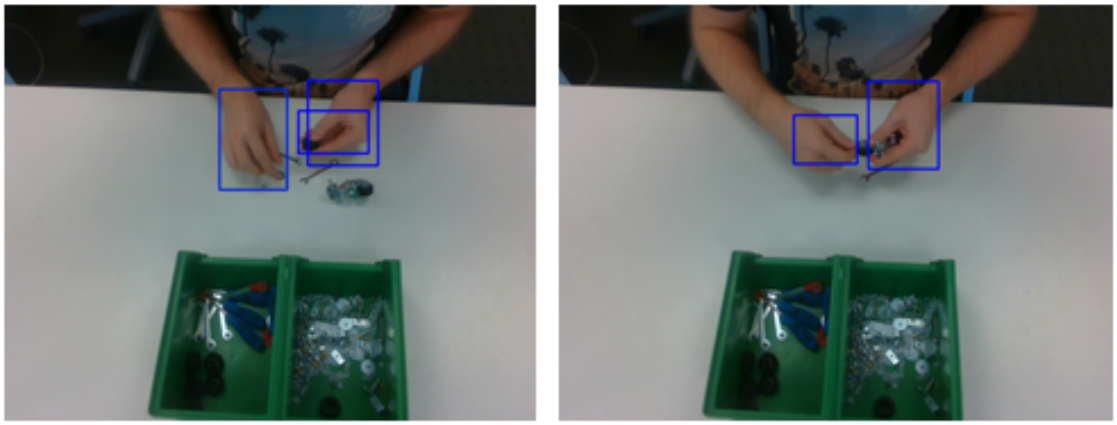


Abbildung 5.6: Fehlerfälle bei der Handsegmentierung. In wenigen Fehlerfällen sind mehr als zwei Handregionen erkannt worden. Für die Segmentierung sind deshalb die zwei größten erkannten Regionen ausgewählt worden. Ein weiteren Fehlerfall stellten zu kleine erkannte Handregionen dar, wie in der rechten Abbildung dargestellt.

Abbildung 5.7: Fehlerfälle wenn sich die Hände nah beieinander befinden. Bei diesen Fällen ist oft eine Region zurückgeben worden oder zurückgegebene Handregionen enthielten noch Teile der anderen Hand, wie in der rechten Abbildung dargestellt.

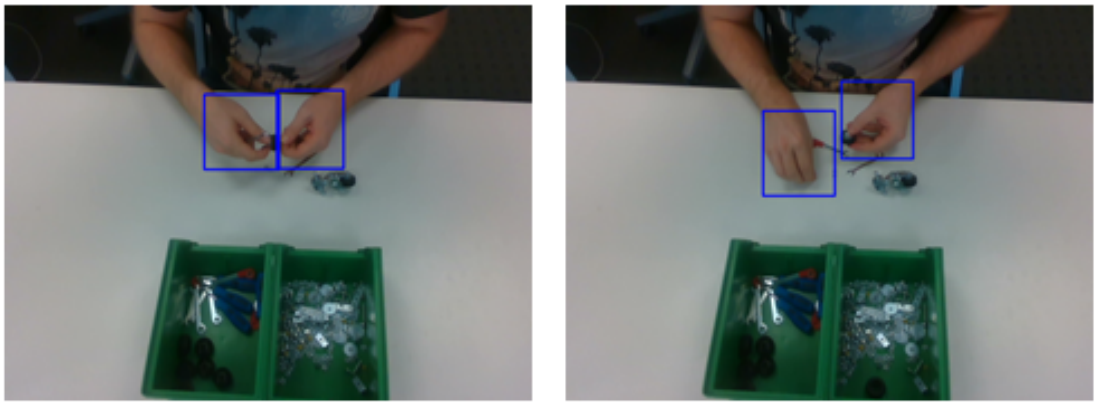


Abbildung 5.8: Korrekt erkannte Handregionen im Testszenario. Die Handsegmentierung funktionierte größtenteils zuverlässig und gibt beide Handregionen als Ergebnis zurück.

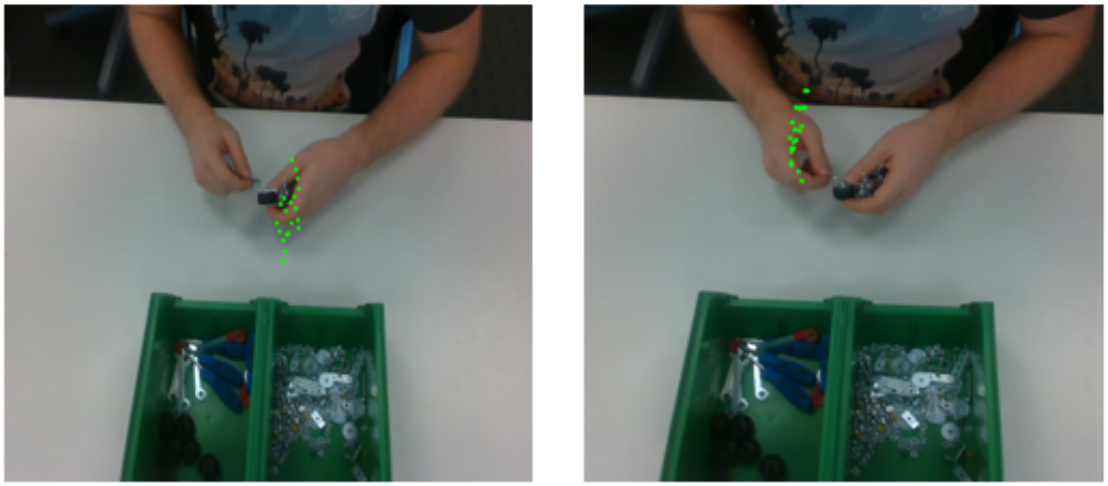


Abbildung 5.9: Für das gewählte Testszenario liefert die Handposenerkennung bei allen getesteten Verfahren keine guten Resultate. Vor allem wenn die Hände sich nah zur Tischplatte befanden, sind die zurückgegebenen Gelenkpositionen der Hand nicht annähernd korrekt. Zur besseren Übersicht sind nur die Gelenkpunktkoordinaten einer Hand eingezeichnet.

6 Diskussion

In diesem Kapitel werden die erzielten Ergebnisse untersucht und mögliche Gründe für die Resultate diskutiert. Des Weiteren werden alternative Ansätze für das Ableiten der Handgelenkkoordinaten vorgestellt und am Ende eine Zusammenfassung der Ergebnisse und mögliche weitere Schritte präsentiert.

6.1 Ergebnisse

Im Folgenden werden die Ergebnisse bei der Handsegmentierung und der Handposenerkennung analysiert.

6.1.1 Ergebnisse Handsegmentierung

Die Ergebnisse in der Handsegmentierung fallen durchaus positiv aus. Nur wenn die Hände nah beieinander sind kommt es zu Ungenauigkeiten bei der Erkennung. Ein Grund hierfür könnte der Datensatz sein, auf dem trainiert worden ist. In diesem ist fast ausschließlich eine einzige Hand im Trainingsbild vorhanden. Weiterhin interagieren die Hände nicht miteinander oder mit anderen Objekten. Das Testszenario ist daher wesentlich komplexer und anspruchsvoller als der verwendete Trainingssatz. Um die Handregionen zuverlässig und robust in einem Montagszenario zu erkennen, sollte deshalb ein Trainingsdatensatz genutzt werden, welcher die Interaktion von den Händen mit anderen Objekten wie Werkzeug oder auch Bauteilen besser repräsentieren kann. Da für dieses Szenario noch kein Datensatz existiert sollte dieser selbst angelegt werden.

In dem Testszenario ist ein Sensor für die Aufnahme des Bildes verwendet worden. Die Verwendung mehrerer Sensoren auf verschiedenen Positionen und Blickwinkeln sollte die Genauigkeit in der Erkennung ebenfalls steigern können. Wird beispielsweise auf dem Testbild des einen Sensors nur eine der zwei Handregionen erkannt, könnten auf dem Bild des anderen Sensors, aufgrund des besseren Blickwinkels, beide Handregionen erkannt werden.

Die hier vorgestellte Methode zur Handsegmentierung ist durchaus in einem Szenario in der manuellen Montage anwendbar. Um die Genauigkeit noch weiter zu steigern sollte die Methode mit einem passenderen Datensatz trainiert werden, um in Fällen, in denen die Hände sich nah beieinander befinden, zuverlässig die korrekten Handregionen zurückzugeben. Eine weiterer Schritt wäre zusätzlich Objekte wie Werkzeuge und Bauteile erkennen zu können, um somit mehr Input für das Assistenzsystem liefern zu können und gegebenenfalls diese aus dem Bild zu entfernen für eine bessere Genauigkeit.

6.1.2 Ergebnisse Handposenerkennung

Die Ergebnisse der Handposenerkennung auf dem Testszenario fallen schlechter aus als erwartet. Während auf dem Testdatensatz noch genaue Handgelenkpunktpositionen bestimmt werden konnten, ist das Ergebnis im Testszenario um ein vielfaches schlechter ausgefallen. Ein Grund dafür könnte die Diskrepanz zwischen den extrahierten Tiefenbildern aus dem Trainingsdatensatz und dem Testszenario sein. In Abbildung 6.1 ist ein Beispiel dargestellt. Im Testdatensatz befinden sich nah um die Hand herum keine anderen Objekte. Im Testszenario dagegen, ist die Tischplatte, Werkzeug und Bauteile in den meisten Fällen nah an den Händen. Deshalb unterscheidet sich das extrahierte Tiefenbild deutlich vom Testdatensatz. Ein weiterer Grund für die schlechten Ergebnisse könnten die Fälle darstellen,in denen sich in einem Tiefenbild der Handregion zusätzlich kleine Bereiche der anderen Hand befinden. Da solche Bilder sich nicht im Trainingsdatensatz befanden, könnte dies eine der Ursachen sein, warum die Verfahren in diesen Fällen schlechtere Ergebnisse liefern.

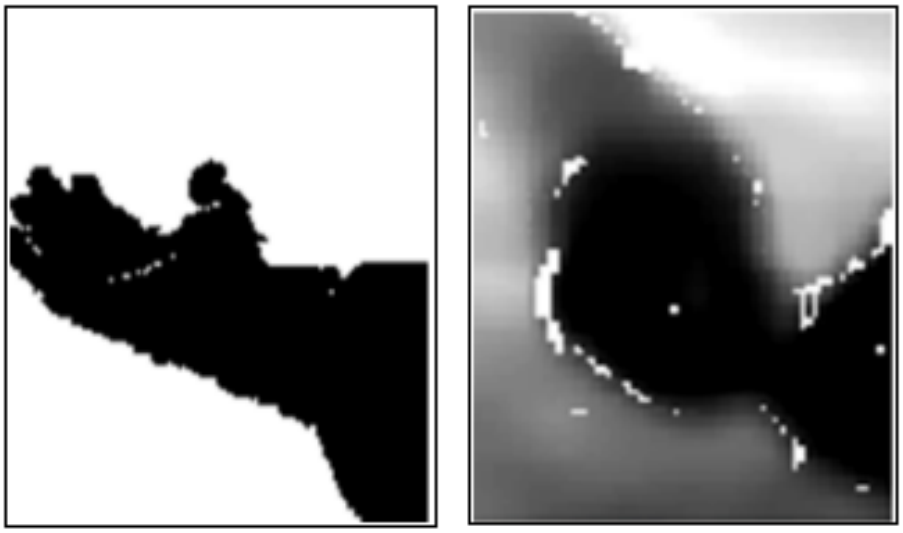


Abbildung 6.1: Vergleich der extrahierten Tiefenbilder aus dem Testdatensatz und dem Testszenario. Pixel die sich näher zum Sensor befinden werden dunkler und Pixel die weiter weg sind heller dargestellt. Auf der linken Seite ist ein Tiefenbild der extrahierten Handregion aus dem BigHand2.2M Datensatz. Rechts befindet sich ein extrahiertes Tiefenbild der Handregion aus dem Testszenario. Im Testszenario befinden sich die Hände viel näher zu anderen Objekten wie der Tischplatte oder Werkzeugen und Bauteilen.

Die erzielten Ergebnisse der Handposenerkennung im Testszenario zeigen deutlich, dass mit dem BigHand2.2M Datensatz und den vorgestellten Verfahren keine zuverlässige, robuste und genaue Handgelenkpunkterkennung in der Montage durchgeführt werden kann. In dem Szenario eines Assistenzsystems für die manuelle Montage sind die erzielten Ergebnisse zu ungenau und könnten nicht verwendet werden um zuverlässige Rückschlüsse auf die Handgelenkpunktpositionen zu erhalten. Bereits in dem simplem Testszenario, indem wenige Bauteile zusammengeschraubt worden sind, waren die Ergebnisse keinerlei ausreichend für eine genaue Bestimmung der Handgelenkpositionen. Die Handposen die in Verbindung mit Werkzeug und Bauteil auftreten, unterscheiden

sich zu sehr von den verfügbaren Posen in den Handposendatensätzen. Um zu entscheiden, ob die hier angewendeten Verfahren in einem solchen Szenario verwendet werden können, müssten diese nochmal mit einem passenden Datensatz trainiert und evaluiert werden.

6.2 Alternativen

Eine alternative Herangehensweise um ein bessere Ergebnis im Montage-Szenario zu erzielen, könnte die Nutzung der statischen Gegebenheiten im der Montageumgebung sein. Befanden sich die Hände im Testszenario nah an der Tischplatte, sorgte dies für ein deutlich unterschiedliches Tiefenbild im Gegensatz zu den Trainingsbildern, wo sich nichts in der näheren Umgebung der Handregion befand. Da die Sensorposition bekannt ist, könnten statische Objekte. wie beispielsweise der Tisch im Testszenario, aus dem Tiefenbild herausgefiltert werden. Werkzeuge und Bauteile könnten auf diese Weise nicht entfernt werden, da ihre Position dynamisch ist und sich öfters ändern kann, vor allem wenn Werkzeuge und Bauteile in den Händen gehalten werden. Die Entfernung der statischen Objekte aus dem Tiefenbild, sollte zu Bildern führen, welche ähnlicher zum dem Trainingsdatensatz sind und somit die Genauigkeit in der Erkennung der Handgelenkpunkte steigern können. Eine weitere Lösung wäre es, einen eigenen Datensatz in einem Montageszenario zu erstellen und die vorgestellten Verfahren mit diesem zu Trainieren. Hierfür müsste eine geeignete Methode entwickelt werden um die Handgelenkpunkte zu tracken und die Wahrheitswerte zu erhalten. Die Schwierigkeit liegt hierbei die Handgelenkpunkte zu finden, auch wenn diese durch Objekte wie Werkzeuge oder Bauteile verdeckt werden.

Eine weitere Anpassung, die für das Testszenario vorgenommen werden könnte, ist die Verwendung mehrerer Sensoren. Durch mehrere Blickwinkel sollte eine genauere Bestimmung der Gelenkpunkte möglich sein. Beispielsweise könnten drei Sensoren verwendetet werden und der Mittelwert aller Handgelenkpunkte aus den drei Tiefenbildern zurückgegeben werden. Dies sollte die Genauigkeit der Erkennung steigern können. Die Laufzeit der Verfahren hingegen würde bei dieser Methode um ein dreifaches ansteigen, da drei Tiefenbilder untersucht werden müssen. Ein Vorteil bei der Verwendung mehrerer Sensoren ist, dass der Bereich in dem die Erkennung stattfinden soll, deutlich vergrößert werden kann.

Ein anderer Ansatz um die Genauigkeit zu steigern, könnte die Kombination mehrerer Verfahren miteinander sein. Auf dem Testdatensatz konnten auf der x-, y- und z-Ebene unterschiedliche Genauigkeiten erzielt werden. Beispielsweise erzielt die Methode DeepPrior++ die besten Ergebnisse auf der z-Achse, die selbst entwickelte Methode ist hingegen auf der x- und y-Ebene genauer gewesen. Ein Ansatz könnte deswegen sein, mehrere Methoden zu verwenden und die x-,y- und z-Koordinaten jeweils aus dem Verfahren als Ergebnis zu Verwenden, das auf der jeweiligen Ebene die höchste Genauigkeit erzielen konnte. Auf diese Weise könnte die Genauigkeit gesteigert werden. Die Performance würde dadurch aber sinken, da mehrere Methoden gleichzeitig verwendet werden müssten.

Ein komplett anderer Ansatz könnte die Verwendung von modellbasierten Methoden sein. Diese sollten aufgrund des verwendeten Handmodells auf ungesehenen Posen und Handposen mit Verdeckung von Gelenkpunkten bessere und robustere Ergebnisse erzielen können [MBS+17]. Da sich jede Hand in Größe und Form von Mensch zu Mensch unterscheidet, müsste die Verfahren aber für jeden Arbeiter in der Montage abgestimmt werden. Um dieses Problem zu umgehen, könnte ein Ansatz aus den hybriden Verfahren verwendet werden. Eine datenbasierte Methoden bestimmt dabei

die Größe der Hand des Nutzers. Beispielsweise könnte ein Arbeiter in der Montage für den ersten Start die Hand eine Sekunde vor den Sensor halten, daraufhin wird die Handgröße bestimmt und die Parameter des verwendeten Handmodells angepasst. Auf diese Weise kann das System automatisch und ohne Aufwand für jeden Nutzer kalibriert werden. Ein solches System sollte ebenfalls auf einem Testszenario ausgewertet werden, um zu entscheiden ob es in dem Anwendungsgebiet der manuellen Montage eingesetzt werden kann.

6.3 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit sind mehrere Arten von Handerkennungsalgorithmen verglichen und vorgestellt worden. Dabei ist ein Überblick über den aktuellen Stand der Technik, der verwendeten Sensorik und den verfügbaren Datensätzen gegeben worden. Für die Auswertung sind drei datenbasierte Methoden und eine selbst entwickelte Methode, sowohl für die Handsegmentierung als auch die Handposenerkennung, implementiert und evaluiert worden. Im Speziellen die Handsegmentierung musste angepasst werden, da aktuelle Verfahren nur eine Hand erkennen und zu strikte Anforderungen an die aufzunehmende Szene stellen, um in einem Montageszenario angewendet werden zu können. Ein großer Fokus der Arbeit war die Auswertung der Einsetzbarkeit solcher Verfahren im einem Montageszenario. Für die Auswertung sind die Verfahren deshalb auf einem Testdatensatz und einem selbst erstellten Testszenario für die manuelle Montage evaluiert worden. Das Testszenario besteht aus einer simplen Montage mehrere Bauteile auf einer Werkbank mit Verwendung von Werkzeugen. Bei der Evaluation ist die Handsegmentierung und die Handposenerkennung der Verfahren getrennt betrachtet worden, da ohne korrekt segmentierte Handregion keine Handposenerkennung durchgeführt werden kann.

Die selbst entwickelte Handsegmentierung konnte größtenteils beide Handregionen im Testszenario korrekt angeben. Probleme und Ungenauigkeiten sind nur aufgetreten, wenn sich die Hände sehr nah beieinander befanden. Hier ist manchmal nur eine der beiden Hände erkannt worden oder die zurückgegebene Handregion hat beide Hände umfasst. Grundsätzlich war die Erkennung für das Testszenario robust und genau genug um in einem solchem Szenario eingesetzt werden zu können.

Die Ergebnisse der Handerkennungsalgorithmen auf dem Testdatensatz sind gut ausgefallen. Die mittlere Abweichung lag bei wenigen Millimetern und die Handpose konnte relativ nah zum Wahrheitswert bestimmt werden. Die selbst entwickelte Methode konnte dabei im Vergleich zu den anderen Verfahren, die besten Ergebnisse auf der x- und y-Ebene erzielen. Auf der z-Achse hingegen, erzielten die anderen verglichenen Handerkennungsalgorithmen bessere Ergebnisse. Bei der Auswertung auf dem Testdatensatz kam hervor, dass die Gelenkpunkte an den Randregionen der Hand am ungenausten bestimmt werden konnten. Dies ist auf die Eigenschaften der Tiefenbilder in den Randregionen zurückzuführen. Die Konturen sind in diesen Regionen deutlich ungenauer zu erkennen als im Zentrum der Hand. Deshalb erzielten die Gelenkpunkte nah zu Mitte der Hand auch die genauesten Ergebnisse.

Innerhalb des Testszenarios konnte keines der Verfahren eine annähernd ausreichende Genauigkeit liefern. Dies ist auf die stark abweichenden Tiefenbilder aus dem Trainingsdatensatz und den erhaltenen Tiefenbildern im Testszenario zurückzuführen. Im Trainingsdatensatz befanden sich in der näheren Umgebung der Handregionen keine anderen Objekte, während im Testszenario die Werkzeuge, Bauteile und auch die Werkbank nah an den Handregionen lagen und zusätzlich auch Handgelenkpunkte verdeckt haben. Allgemein ergab sich durch die Evaluation, dass die hier

getesteten Verfahren mit den aktuell verfügbaren Datenätzen keine ausreichende Genauigkeit in einem Montageszenario erzielen und deshalb in diesem Bereich nicht zuverlässig eingesetzt werden können. Um eine endgültige Aussage zu treffen, ob die aktuellen Handerkennungsalgorithmen für ein Montageszenario verwendet werden können, sollten diese auf einem Datensatz trainiert werden, welcher ebenfalls aus Basis eines Montageszenario erstellt worden ist.

Ein nächster Schritt, basierend auf den hier erzielten Ergebnissen, könnte deshalb die Erstellung eines eigenen Datensatzes sein. Auf diese Weise können die Handerkennungsalgorithmen mit einem realitätsnäherem Datensatz zum Montageszenario neu trainiert und ausgewertet werden. Weiterhin könnten alternative Ansätze, wie im Abschnitt ”Alternativen” beschrieben, getestet werden. Die hier erzielten Ergebnisse legen dar, dass der aktuelle Stand der Handerkennungsalgorithmen nur für simple Handposen und eingeschränkte Szenarios eine ausreichende Genauigkeit erzielen kann. Die Methoden sind für die Erkennung einer einzelnen Handregion ausgelegt. Verdeckte Gelenkpunkte und Objekte in der Nähe der Handregion verschlechtern die Genauigkeit der Erkennung deutlich. Für komplexe Anwendungsszenarios, wie die manuelle Montage, existieren aktuell keine öffentlich verfügbaren Datensätze und müssen deshalb erst angelegt werden.

Die Entwicklung der Handerkennungsalgorithmen ist bei Weitem noch nicht abgeschlossen und auch zukünftig werden weitere Verfahren entwickelt und vorgestellt werden. Vor allem für komplexere Anwendungsszenarios müssen die aktuellen Verfahren verbessert und erweitert werden.

Literaturverzeichnis

[AFG14] C. Amon, F. Fuhrmann, F. Graf. „Evaluation of the spatial resolution accuracy of the face tracking system for kinect for windows v1 and v2“. In: *Proceedings of the 6th Congress of the Alps Adria Acoustics Association*. 2014, S. 16–17 (zitiert auf S. 22).

[Asu18] Asus. *XtionPRO*. 2018. URL: https://www.asus.com/3D-Sensor/Xtion%5C_PRO/specifications/ (besucht am 09. 05. 2018) (zitiert auf S. 22).

[CP11] T. I. Cerlinca, S. G. Pentiuc. „Robust 3D hand detection for gestures recognition“. In: *Intelligent Distributed Computing V*. Springer, 2011, S. 259–264 (zitiert auf S. 24).

[CS2] CS231n. *Convolutional Neural Networks*. URL: http://cs231n.github.io/convolutional-networks/ (zitiert auf S. 13).

[CTAW11] M. J. Choi, V. Y. Tan, A. Anandkumar, A. S. Willsky. „Learning latent tree graphical models“. In: *Journal of Machine Learning Research* 12.May (2011), S. 1771–1812 (zitiert auf S. 31).

[CYL16] H. Cheng, L. Yang, Z. Liu. „Survey on 3D Hand Gesture Recognition.“ In: *IEEE Trans. Circuits Syst. Video Techn.* 26.9 (2016), S. 1659–1673 (zitiert auf S. 19).

[Din15] M. Ding. „Human motion analysis: From gait modeling to shape representation and pose estimation“. Diss. Oklahoma State University, 2015 (zitiert auf S. 17).

[EBN+07] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, X. Twombly. „Vision-based hand pose estimation: A review“. In: *Computer Vision and Image Understanding* 108.1-2 (2007), S. 52–73 (zitiert auf S. 27, 30).

[Fol16] lostleaves Follow. *Deep Learning behind Prisma*. 2016. URL: https://www.slideshare.net/lostleaves/deep-learning-behind-prisma-66647472 (zitiert auf S. 12).

[FVR+17] S. R. Fanello, J. Valentin, C. Rhemann, A. Kowdle, V. Tankovich, P. Davidson, S. Izadi. „UltraStereo: Efficient Learning-based Matching for Active Stereo Systems“. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, S. 6535–6544 (zitiert auf S. 21).

[GLYT16] L. Ge, H. Liang, J. Yuan, D. Thalmann. „Robust 3d hand pose estimation in single depth images: from single-view cnn to multi-view cnns“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, S. 3593–3601 (zitiert auf S. 33, 52).

[GLYT17] L. Ge, H. Liang, J. Yuan, D. Thalmann. „3d convolutional neural networks for efficient and robust hand pose estimation from single depth images“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Bd. 1. 2017, S. 5 (zitiert auf S. 35).

[GOC17] F. Gomez-Donoso, S. Orts-Escolano, M. Cazorla. „Large-scale multiview 3d hand pose dataset“. In: *arXiv preprint arXiv:1707.03742* (2017) (zitiert auf S. 43).

[GWC+17] H. Guo, G. Wang, X. Chen, C. Zhang, F. Qiao, H. Yang. „Region ensemble network: Improving convolutional network for hand pose estimation“. In: *Image Processing (ICIP), 2017 IEEE International Conference on*. IEEE. 2017, S. 4512–4516 (zitiert auf S. 25, 34, 38, 52).

[HRS+17] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama et al. „Speed/accuracy trade-offs for modern convolutional object detectors“. In: *IEEE CVPR*. 2017 (zitiert auf S. 24).

[HZRS16] K. He, X. Zhang, S. Ren, J. Sun. „Deep residual learning for image recognition“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, S. 770–778 (zitiert auf S. 14, 15, 33, 45).

[HZX+18] C. Hong, Z. Zeng, R. Xie, W. Zhuang, X. Wang. „Domain adaptation with low-rank alignment for weakly supervised hand pose recovery“. In: *Signal Processing* 142 (2018), S. 223–230 (zitiert auf S. 25).

[Int18] Intel. *Intel RealSense D435 Specifications*. 2018. URL: https://click.intel.com/intelr-realsensetm-depth-camera-d435.html (besucht am 09. 05. 2018) (zitiert auf S. 22, 41).

[KKA14] F. Kirac, Y. E. Kara, L. Akarun. „Hierarchically constrained 3D hand pose estimation using regression forests from single frame depth data“. In: *Pattern Recognition Letters* 50 (2014), S. 91–100 (zitiert auf S. 30).

[KKKA12] C. Keskin, F. Kıraç, Y. E. Kara, L. Akarun. „Hand pose estimation and hand shape classification using multi-layered randomized decision forests“. In: *European Conference on Computer Vision*. Springer. 2012, S. 852–863 (zitiert auf S. 30, 31).

[Kon10] K. Konolige. „Projected texture stereo“. In: *2010 IEEE International Conference on Robotics and Automation* (2010), S. 148–155 (zitiert auf S. 22).

[KRSB18] S. Khan, H. Rahmani, S. A. A. Shah, M. Bennamoun. *A guide to convolutional neural networks for computer vision*. Bd. 8. 1. Morgan & Claypool Publishers, 2018, S. 1–207. ISBN: 1-68173-022-7 (zitiert auf S. 11–15).

[LFP11] M. de La Gorce, D. J. Fleet, N. Paragios. „Model-based 3d hand pose estimation from monocular video“. In: *IEEE transactions on pattern analysis and machine intelligence* 33.9 (2011), S. 1793–1805 (zitiert auf S. 27).

[LJ09] Z. Li, R. Jarvis. „Real time hand gesture recognition using a range camera“. In: *Australasian Conference on Robotics and Automation*. 2009, S. 21–27 (zitiert auf S. 24).

[LJC06] X. Lu, A. K. Jain, D. Colbry. „Matching 2.5 D face scans to 3D models“. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 1 (2006), S. 31–43 (zitiert auf S. 17).

[LYT12] H. Liang, J. Yuan, D. Thalmann. „Hand pose estimation by combining fingertip tracking and articulated ICP“. In: *Proceedings of the 11th acm siggraph international conference on virtual-reality continuum and its applications in industry*. ACM. 2012, S. 87–90 (zitiert auf S. 28).

[Mat] Mathworks. *Introduction to Deep Learning: What Are Convolutional Neural Networks? Video*. URL: https://de.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html (zitiert auf S. 14).

[MBS+17] F. Mueller, F. Bernard, O. Sotnychenko, D. Mehta, S. Sridhar, D. Casas, C. Theobalt. „GANerated Hands for Real-time 3D Hand Tracking from Monocular RGB“. In: *arXiv preprint arXiv:1712.01057* (2017) (zitiert auf S. 65).

[MCL17] G. Moon, J. Y. Chang, K. M. Lee. „V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation from a Single Depth Map“. In: *arXiv preprint arXiv:1711.07399* (2017) (zitiert auf S. 26, 35, 36, 38, 50, 51).

[Mel] Melonwan. *Github Repository melonwan/denseReg*. URL: https://github.com/melonwan/denseReg (zitiert auf S. 49).

[MES17] J. Malik, A. Elhayek, D. Stricker. „Simultaneous Hand Pose and Skeleton Bone-Lengths Estimation from a Single Depth Image“. In: *arXiv preprint arXiv:1712.03121* (2017) (zitiert auf S. 36).

[mks] mks0601. *Github Repository mks0601/V2V-PoseNet_RELEASE*. URL: https://github.com/mks0601/V2V-PoseNet%5C_RELEASE (zitiert auf S. 51, 57).

[MN06] Z. Mo, U. Neumann. „Real-time hand pose recognition using low-resolution depth images“. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Bd. 2. IEEE. 2006, S. 1499–1505 (zitiert auf S. 23).

[MSFM10] L. Maier-Hein, T. R. dos Santos, A. M. Franz, H.-P. Meinzer. „Iterative Closest Point Algorithm in the Presence of Anisotropic Noise.“ In: *Bildverarbeitung für die Medizin 2010* (2010), S. 231–235 (zitiert auf S. 28).

[MZ15] D. Mohr, G. Zachmann. „Hand Pose Recognition—Overview and Current Research“. In: *Virtual Realities*. Springer, 2015, S. 108–129 (zitiert auf S. 9, 30).

[NWNT17] N. Neverova, C. Wolf, F. Nebout, G. W. Taylor. „Hand pose estimation through semi-supervised and weakly-supervised learning“. In: *Computer Vision and Image Understanding* 164 (2017), S. 56–67 (zitiert auf S. 19).

[NYD16] A. Newell, K. Yang, J. Deng. „Stacked hourglass networks for human pose estimation“. In: *European Conference on Computer Vision*. Springer. 2016, S. 483–499 (zitiert auf S. 35, 36, 46, 48).

[Obe] Oberweger. *Github Repository DeepPrior++*. URL: https://github.com/moberweger/deep-prior-pp (zitiert auf S. 46, 55).

[OKA11] I. Oikonomidis, N. Kyriazis, A. A. Argyros. „Efficient model-based 3D tracking of hand articulations using Kinect.“ In: *BmVC*. Bd. 1. 2. 2011, S. 3 (zitiert auf S. 28).

[OL17] M. Oberweger, V. Lepetit. „Deepprior++: Improving fast and accurate 3d hand pose estimation“. In: *ICCV workshop*. Bd. 840. 2017, S. 2 (zitiert auf S. 33, 34, 38, 42, 43, 45, 46).

[Onp] Onpulson. *Heatmap*. URL: https://www.onpulson.de/lexikon/heatmap/ (zitiert auf S. 17).

[OWL15] M. Oberweger, P. Wohlhart, V. Lepetit. „Hands deep in deep learning for hand pose estimation“. In: *arXiv preprint arXiv:1502.06807* (2015) (zitiert auf S. 31–33, 38).

[pra] prashanth. *Option to select between to different kind of depth images*. URL: http://official-rtab-map-forum.67519.x6.nabble.com/Option-to-select-between-to-different-kind-of-depth-images-td2510.html (zitiert auf S. 16, 19).

[QSW+14] C. Qian, X. Sun, Y. Wei, X. Tang, J. Sun. „Realtime and robust hand tracking from depth“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, S. 1106–1113 (zitiert auf S. 27, 28).

[RF18] J. Redmon, A. Farhadi. „Yolov3: An incremental improvement“. In: *arXiv preprint arXiv:1804.02767* (2018) (zitiert auf S. 43, 44).

[RHGS15] S. Ren, K. He, R. Girshick, J. Sun. „Faster r-cnn: Towards real-time object detection with region proposal networks“. In: *Advances in neural information processing systems*. 2015, S. 91–99 (zitiert auf S. 24).

[san] sanko. *ICP*. URL: http://www.sanko-shoko.net/note.php?id=3c5m (zitiert auf S. 28).

[SFC+11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, A. Blake. „Real-time human pose recognition in parts from single depth images“. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. Ieee. 2011, S. 1297–1304 (zitiert auf S. 24).

[SH] J. Schöning, G. Heidemann. „Taxonomy of 3D sensors“. In: *Argos* 3 (), P100 (zitiert auf S. 20).

[SKR+15] T. Sharp, C. Keskin, D. Robertson, J. Taylor, J. Shotton, D. Kim, C. Rhemann, I. Leichter, A. Vinnikov, Y. Wei et al. „Accurate, robust, and flexible real-time hand tracking“. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM. 2015, S. 3633–3642 (zitiert auf S. 25, 36).

[SLK15] H. Sarbolandi, D. Lefloch, A. Kolb. „Kinect range sensing: Structured-light versus time-of-flight kinect“. In: *Computer vision and image understanding* 139 (2015), S. 1–20 (zitiert auf S. 19, 20).

[SM12] J. Suarez, R. R. Murphy. „Hand gesture recognition with depth images: A review“. In: *Ro-man, 2012 IEEE*. IEEE. 2012, S. 411–417 (zitiert auf S. 24).

[Spa] C. Spark. *Deep learning for complete beginners: convolutional neural networks with keras*. URL: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html (zitiert auf S. 12).

[SRY+15] J. S. Supancic III, G. Rogez, Y. Yang, J. Shotton, D. Ramanan. „Depthbased hand pose estimation: methods, data, and challenges. arXiv preprint“. In: *arXiv preprint arXiv:1504.06378* (2015) (zitiert auf S. 31).

[SSH+17] J. Sanchez-Riera, K. Srinivasan, K.-L. Hua, W.-H. Cheng, M. A. Hossain, M. F. Alhamid. „Robust rgb-d hand tracking using deep learning priors“. In: *IEEE Transactions on Circuits and Systems for Video Technology* (2017) (zitiert auf S. 36).

[SSN+14] E. Stergiopoulou, K. Sgouropoulos, N. Nikolaou, N. Papamarkos, N. Mitianoudis. „Real time hand detection in a complex background“. In: *Engineering Applications of Artificial Intelligence* 35 (2014), S. 54–70 (zitiert auf S. 23).

[SWL+15] X. Sun, Y. Wei, S. Liang, X. Tang, J. Sun. „Cascaded hand pose regression“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, S. 824–832 (zitiert auf S. 33).

[Tan11] M. Tang. „Recognizing hand gestures with microsoft's kinect“. In: *Palo Alto: Department of Electrical Engineering of Stanford University:[sn]* (2011) (zitiert auf S. 23).

[Ter] Terabee. *Time-of-Flight principle, measuring the distance between sensor/object*. URL: https://www.terabee.com/time-of-flight-principle/ (zitiert auf S. 21).

[TJTK14] D. Tang, H. Jin Chang, A. Tejani, T.-K. Kim. „Latent regression forest: Structured estimation of 3d articulated hand posture“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, S. 3786–3793 (zitiert auf S. 30, 31, 38).

[TSLP14] J. Tompson, M. Stein, Y. Lecun, K. Perlin. „Real-time continuous pose recovery of human hands using convolutional networks“. In: *ACM Transactions on Graphics (ToG)* 33.5 (2014), S. 169 (zitiert auf S. 36).

[TYK13] D. Tang, T.-H. Yu, T.-K. Kim. „Real-time articulated hand pose estimation using semi-supervised transductive regression forests“. In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, S. 3224–3231 (zitiert auf S. 30).

[WFOY18] X. Wu, D. Finnegan, E. O'Neill, Y.-L. Yang. „HandMap: Robust Hand Pose Estimation via Intermediate Dense Guidance Map Supervision“. In: *15th European Conference on Computer Vision*. 2018 (zitiert auf S. 17).

[wik] wikibooks. *Artificial Neural Networks/Neural Network Basics*. URL: https://en.wikibooks.org/wiki/Artificial%5C_Neural%5C_Networks/Neural%5C_Network%5C_Basics (zitiert auf S. 10).

[WP09] R. Y. Wang, J. Popović. „Real-time hand-tracking with a color glove“. In: *ACM transactions on graphics (TOG)*. Bd. 28. 3. ACM. 2009, S. 63 (zitiert auf S. 18).

[WPVY17] C. Wan, T. Probst, L. Van Gool, A. Yao. „Dense 3D Regression for Hand Pose Estimation“. In: *arXiv preprint arXiv:1711.08996* (2017) (zitiert auf S. 35, 38, 47, 48).

[XC13] C. Xu, L. Cheng. „Efficient hand pose estimation from a single depth image“. In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, S. 3456–3462. DOI: 10.1007/s11263-015-0826-9 (zitiert auf S. 30).

[XGD+17] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He. „Aggregated residual transformations for deep neural networks“. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, S. 5987–5995 (zitiert auf S. 51, 52).

[YGS+18] S. Yuan, G. Garcia-Hernando, B. Stenger, G. Moon, J. Yong Chang, K. Mu Lee, P. Molchanov, J. Kautz, S. Honari, L. Ge et al. „Depth-based 3d hand pose estimation: From current achievements to future goals“. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (zitiert auf S. 57, 58).

[YYGK17] S. Yuan, Q. Ye, G. Garcia-Hernando, T.-K. Kim. „The 2017 hands in the million challenge on 3d hand pose estimation“. In: *arXiv preprint arXiv:1707.02237* (2017) (zitiert auf S. 18, 36, 39).

[YYS+17] S. Yuan, Q. Ye, B. Stenger, S. Jain, T.-K. Kim. „Bighand2. 2m benchmark: Hand pose dataset and state of the art analysis“. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, S. 2605–2613 (zitiert auf S. 25–27, 56).

[ZBA09] X. Zabulis, H. Baltzakis, A. A. Argyros. „Vision-Based Hand Gesture Recognition for Human-Computer Interaction.“ In: *The universal access handbook* 34 (2009), S. 30 (zitiert auf S. 23).

[Zha12] Z. Zhang. „Microsoft kinect sensor and its effect“. In: *IEEE multimedia* 19.2 (2012), S. 4–10 (zitiert auf S. 19).

[ZWZ+16] X. Zhou, Q. Wan, W. Zhang, X. Xue, Y. Wei. „Model-based deep hand pose estimation“. In: *arXiv preprint arXiv:1606.06854* (2016) (zitiert auf S. 29, 38).

Alle URLs wurden zuletzt am 20. 09. 2018 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift