

UNIVERSITÄT BREMEN

BACHELORARBEIT

---

**Einfluss verschiedener Handmodelle auf  
gerenderte Trainingsdaten zur  
Anwendung in der Posenbestimmung von  
Objekten**

---

*Autor:*  
Max MIEHE  
4578043

*Betreuer:*  
Janis ROSSKAMP

1. Gutachter: Prof. Gabriel ZACHMANN  
2. Gutachter: Dr. Tim LAUE

*Eine Abschlussarbeit zum Erlangen  
des wissenschaftlichen Grades Bachelor of Science*

*in*

Computer Graphics and Virtual Reality Research Lab  
Fachbereich 3

19. Oktober 2023

# Eidesstattliche Erklärung

## A) Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet. Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht. Die elektronische Fassung der Arbeit stimmt mit der gedruckten Version überein. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

## B) Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:

1. Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2. Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.
3.  Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
  - Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
  - Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

## C) Einverständniserklärung zur Überprüfung der elektronischen Fassung der Bachelorarbeit /Masterarbeit durch Plagiatssoftware

Eingereichte Arbeiten können nach § 18 des Allgemeinen Teil der Bachelor- bzw. der Master- prüfungsordnungen der Universität Bremen mit qualifizierter Software auf Plagiatsvorwürfe untersucht werden. Zum Zweck der Überprüfung auf Plagiate erfolgt das Hochladen auf den Server der von der Universität Bremen aktuell genutzten Plagiatssoftware.

- Ich bin damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum oben genannten Zweck dauerhaft auf dem externen Server der aktuell von der Universität Bremen genutzten Plagiatssoftware, in einer institutioneigenen Bibliothek (Zugriff nur durch die Universität Bremen), gespeichert wird.
- Ich bin nicht damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum o.g. Zweck dauerhaft auf dem externen Server der aktuell von der Universität Bremen genutzten Plagiatssoftware, in einer institutioneigenen Bibliothek (Zugriff nur durch die Universität Bremen), gespeichert wird.

Mit meiner Unterschrift versichere ich, dass ich die obenstehenden Erklärungen gelesen und verstanden habe und bestätige die Richtigkeit der gemachten Angaben.

Signed:

---

Date:

---

UNIVERSITÄT BREMEN

# Zusammenfassung

Faculty Name  
Fachbereich 3

Bachelor of Science

## **Einfluss verschiedener Handmodelle auf gerenderte Trainingsdaten zur Anwendung in der Posenbestimmung von Objekten**

von Max MIEHE

Synthetische Trainingsdaten sind eine etablierte Möglichkeit zur Augmentierung und Substituierung bestehender Bilddatensätze zum Training von Convolutional Neural Networks. Diese Bachelorarbeit erforscht die Optimierungsmöglichkeiten synthetischer Trainingsdaten in der 6D Posenbestimmung für Objekte im Kontext von Hand-Objekt-Interaktionen. Konkret wird der Einfluss drei verschiedener Handmodelle in den Trainingsdaten auf die Erkennungsfähigkeit eines damit trainierten Neuronalen Netzes untersucht. Verglichen wird das einfache MANO Handmodell einerseits mit dem komplexeren NIMBLE Handmodell und andererseits mit einem MANO Modell mit einem simplen Armfortsatz. Zur Konstruktion vergleichbarer Datensätze wurde eine mehrschrittige Pipeline zur Simulation von Hand-Objekt Interaktionen in unterschiedlichen künstlichen Szenen erstellt. Mit den drei entstandenen synthetischen Datensätzen wurden drei Neuronale Netze trainiert und anschließend auf realen Hand-Objekt Interaktionen ausgewertet. Die Ergebnisse zeigen keinen Unterschied zwischen den verwendeten realistischen und weniger realistischen Händen. Für den Datensatz mit Armfortsatz konnte dagegen eine leichte Verbesserung der Ergebnisse festgestellt werden. Die Resultate lassen darauf schließen, dass bei synthetischen Trainingsdaten zur 6D Posenbestimmung von Objekten in Hand Objekt-Interaktion die sichtbaren Gliedmaßen einen Einfluss auf die Erkennungsfähigkeit eines Netzes haben können und berücksichtigt werden sollten, während der Realismus der simulierten Hände in diesem Kontext nur eine untergeordnete Rolle spielt.



## *Danksagung*

Ich möchte zu aller erst Janis, meinem inhaltlichen Betreuer, für seine kontinuierliche Unterstützung danken. Er war jederzeit für Fragen und Anregungen verfügbar und die vielen Gespräche die wir im Laufe der Zeit hatten waren immer produktiv und in einer angenehmen Atmosphäre. Außerdem hat Janis für mich das Training mit meinen Datensätzen und die technische Auswertung übernommen und mir durch seine Erfahrung damit viel Zeit und Kopfschmerzen erspart. Außerdem möchte ich meinen Eltern dafür danken, dass sie mir, trotz dreier Zusatzsemester, ohne Fragen zu stellen ein sorgenfreies Vollzeitstudium ermöglicht haben und mir für den Start ins Erwachsenenleben den Rücken freigehalten haben. Zuletzt gebührt auch ein besonderer Dank all denen, die sich meiner Schachtelsätze, Rechtschreibung und auch Inhaltlichen Auffälligkeiten auf fast 45 Seiten Text erbarmt haben und damit Ordnung in diese Arbeit gebracht haben.

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	<b>i</b>
<b>Zusammenfassung</b>	<b>iii</b>
<b>Danksagung</b>	<b>iv</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Struktur . . . . .	2
<b>2 Stand der Forschung</b>	<b>4</b>
2.1 Maschinelles Lernen . . . . .	4
2.1.1 Überwachtes Lernen . . . . .	5
2.1.2 Neuronale Netze . . . . .	6
2.1.3 Training Neuronaler Netze . . . . .	8
2.1.4 Convolutional Neural Networks . . . . .	9
2.2 Trainingsdaten . . . . .	12
2.2.1 Synthetische Trainingsdaten . . . . .	13
2.3 6D Posenbestimmung von Objekten . . . . .	15
2.3.1 Hand-Objekt Datensätze . . . . .	16
2.4 Handmodelle . . . . .	18
2.4.1 Handposen Tools . . . . .	19
2.5 Inverse Kinematik . . . . .	19
2.5.1 Prinzip, Funktionsweise, Anwendung . . . . .	19
<b>3 Methoden</b>	<b>22</b>
3.1 Die Pipeline . . . . .	22
3.2 Objektdatensatz . . . . .	23
3.3 Markierung der Kontaktpunkte . . . . .	24
3.4 Handmodelle . . . . .	25
3.4.1 MANO . . . . .	26
3.4.2 NIMBLE . . . . .	27
3.5 MinimalIK . . . . .	28
3.5.1 Besonderheiten für NIMBLE . . . . .	29
3.6 Posen Editor . . . . .	30
3.6.1 Funktionen . . . . .	30
3.7 BlenderProc2 . . . . .	31
3.7.1 MANO Vorverarbeitung und Armfortsatz . . . . .	33
3.8 Resultate . . . . .	33

<b>4</b>	<b>Evaluation</b>	<b>35</b>
4.1	Metriken . . . . .	35
4.2	Ergebnisse . . . . .	36
4.2.1	Vergleich der Metriken . . . . .	36
4.2.2	Präzision/Grenzwert Kurven . . . . .	38
4.3	Gegriffene und nicht gegriffene Objekte . . . . .	39
4.3.1	Teilung der Testdaten . . . . .	39
4.3.2	50/50 Netz . . . . .	40
<b>5</b>	<b>Fazit</b>	<b>41</b>
5.1	Grenzen dieser Arbeit . . . . .	41
5.2	Fazit . . . . .	41
5.3	Ausblick . . . . .	42
5.4	Lessons Learned . . . . .	43
5.4.1	Wissenschaftlichkeit . . . . .	43
5.4.2	Ethik . . . . .	43
<b>A</b>	<b>Zusätzliche Bilder</b>	<b>44</b>
<b>B</b>	<b>Vollständige Daten der Evaluation</b>	<b>48</b>
	<b>Literatur</b>	<b>50</b>

# Kapitel 1

## Einleitung

### 1.1 Motivation

Computer sind allgemein bekannt als mächtige *Rechenmaschinen*, die mittlerweile so viele Alltagsaufgaben der Menschen übernehmen können, dass sie aus der funktionierenden Gesellschaft nicht mehr wegzudenken sind. Im Bereich der Bildverarbeitung waren Computer der Komplexität der menschlichen Wahrnehmung aber lange unterlegen. Regelbasierte Algorithmen funktionieren in streng kontrollierten, industriellen Umgebungen zwar gut, stoßen außerhalb davon aber sehr schnell an ihre Grenzen. Mit der Entdeckung des Potenzials von *Deep Learning* und *Neuronalen Netzen* im letzten Jahrzehnt, taten sich auch neue Ansätze für die Bildverarbeitung auf. Computer können heute schon beliebige Bilder mit hoher Genauigkeit semantisch voneinander unterscheiden und klassifizieren. Eine Aufgabe, die aufgrund der unendlich großen Varianz von Bildern in diesem Ausmaß bisher eigentlich nur das menschliche Hirn lösen konnte. *Convolutional Neural Networks* versuchen diese Fähigkeiten nachzuahmen um aus Bildern qualitative oder quantitative Merkmale zu extrahieren. Dabei kann So wie wir Menschen unsere Entscheidungen auf Basis lebenslanger Erfahrung treffen, so lernen Neuronale Netze aus digitalisierten Daten ihr eigenes Verständnis eines Zusammenhangs und können darauf basierend eigene Entscheidungen fällen. Wenn dieser Lernprozess durch den Menschen angestoßen und überwacht wird, nennt man ihn auch *Training*. Die Qualität, Quantität und Varianz der verfügbaren Daten haben entsprechend große Auswirkungen auf den Lernerfolg beim Training eines Neuronalen Netzes. Befindet sich beispielsweise die Zieldomäne einer Bildverarbeitungsaufgabe in der realen Welt, so liegt es erst mal nahe die Maschine auch mit realen Bildern zu trainieren. Die Beschaffung und Annotation von realen Bildern kann allerdings je nach Aufgabe ein sehr aufwendiger Prozess sein.

*Convolutional Neural Networks (CNNs)* erfreuen sich im Bereich der Bildverarbeitung aufgrund ihres Potenzials zwar großer Beliebtheit, ein relevanter Nachteil lernender Algorithmen für die Bildverarbeitung ist allerdings der angesprochene intrinsische Bedarf nach großen Mengen an realistischen, variantenreichen Daten, mit denen gelernt werden kann. Um dem Aufwand der Beschaffung von realen Bilddaten entgegen zu wirken greift man gerne zu synthetischen Daten und tauscht damit häufig Realismus gegen größere Datensätze.

Ich möchte mich in dieser Arbeit mit synthetischen Trainingsdaten für einen kleinen Teilbereich der vielfältigen Anwendungszwecke von *Convolutional Neural Networks* auseinandersetzen. Die 6D Posenbestimmung von Objekten beschäftigt sich mit der Schätzung von vollständigen Objektposes aus sensorischen Tiefendaten, Farbbildern oder einer Kombination daraus. Anwendungen der reinen Objektposesbestimmung sind vor allem autonome Mobilität, die Robotik und Virtual- und Augmented

Reality. Meine Arbeit fokussiert sich auf Trainingsdaten für Netze zur 6D Objekt-  
posenbestimmung bei menschlicher Hand-Objekt Interaktion. Besonders im Bereich  
der VR und AR Anwendungen ist die akkurate Posenbestimmung von Objekten in  
der realen Welt relevant für eine reibungslose Mensch-Computer-Interaktion.

Die Komplexität der menschlichen Hand realistisch zu simulieren ist aufgrund  
der hohen Sensibilität des Menschen gegenüber unnatürlichen Handbewegungen  
eine schwierige Aufgabe. Zur Simulation von menschlichen Händen werden oft  
Handmodelle genutzt. Das Handmodell MANO repräsentiert die Form und Pose  
menschlicher Hände über eine Reihe von Parametern. Das NIMBLE Handmodell  
ist effektiv eine Weiterentwicklung des MANO Modells in Sachen der Komplexität  
und der Textur des resultierenden 3D Modells. Für die sensitive menschliche Wahr-  
nehmung von Händen ist der zunehmende Realismus in Pose und Textur klar er-  
kennbar (Abb. 1.1). Angesichts des Optimierungspotenzials von synthetischen Trai-  
ningsdaten stellt sich hier die Frage, ob der visuelle Unterschied in synthetischen  
Trainingsdaten einen relevanten Einfluss auf die Erkennungsfähigkeit eines Neuro-  
nalen Netzes hat.



ABBILDUNG 1.1: Links der Render einer MANO Hand und rechts  
der Render einer NIMBLE Hand, beide in einer ähnlichen Pose. Das  
MANO Modell wurde von mir eingefärbt.

Die Simulation mit den angesprochenen Handmodellen lässt außerdem den rest-  
lichen menschlichen Körper außer Acht. Daher stellt sich auch die Frage, in wie weit  
die Simulation von Körperteilen, die nicht in direktem Kontakt mit dem Objekt ste-  
hen oder es verdecken, relevant für das Training mit synthetischen Daten ist. Dazu  
erzeuge ich selbst einige unterschiedliche Datensätze und lasse mehrere Instanzen  
eines neuronalen Netzes auf diesen Datensätzen trainieren um diese Effekte unter-  
suchen zu können.

## 1.2 Struktur

Die Arbeit teilt sich in drei Kernteile auf. Ich beginne in Kapitel 2 mit einem Über-  
blick über die Grundlagen des maschinellen Lernens und der Konzepte von Neuro-  
nalen Netzen, so wie über die Bedeutung von Trainingsdaten für die Konstruktion  
lernender Algorithmen. Daraufhin spreche ich über den Stand der Forschung im  
Bereich der 6D Posenbestimmung und der Handmodelle und schließe mit einem

weiteren Grundlagenteil zu relevanten Methoden der Inversen Kinematik zur Optimierung von skelettalen Körpern.

In Kapitel 3 beschreibe ich die Pipeline, mit der ich die synthetischen Datensätze zur Evaluation der Fragestellungen erzeuge. Dazu spreche ich über die Wahl des genutzten Objektdatensatzes, die verwendeten Handmodelle und wie die Objekte mit den Händen in einer eigenen Anwendung zusammengeführt werden. Zuletzt erläutere ich den letzten Schritt der Pipeline in dem die Bilddatensätze mit einer fotorealistischen Render-Engine erzeugt werden und gebe einen Überblick über die von mir erzeugten Datensätze.

In Kapitel 4 präsentiere ich das Setup für die Evaluierung der Datensätze und erkläre die relevanten Metriken. Dann zeige ich die Resultate eines auf den unterschiedlichen Datensätzen trainierten Netzes und werte sie hinsichtlich der Fragestellungen aus.

In Kapitel 5 ziehe ich ein inhaltliches Fazit aus den Ergebnissen und reflektiere über meine Arbeit.

## Kapitel 2

# Stand der Forschung

In diesem Kapitel möchte ich mich zuerst einmal mit den Grundlagen des maschinellen Lernens befassen. Dabei fokussiere ich mich vor allem auf CNNs im Kontext der Neuronalen Netze und wie diese Methoden aus Beispielen in Form von Trainingsdaten das Bewältigen einer Aufgabe lernen können. Um meine Arbeit in dieses umfangreiche Forschungsgebiet einzuordnen, beschäftige ich mich daraufhin konkret mit dem Konzept von synthetischen Trainingsdaten und spreche über relevante Methoden und Datensätze zur 6D-Posenbestimmung von Objekten. Zuletzt gebe ich noch einen Überblick über zwei Konzepte abseits des maschinellen Lernens, die aber im Kontext meiner Arbeit relevant sind. Zum Einen geht es dort um die Techniken und Entwicklungen in Bezug auf Handmodelle und zum Anderen geht es um das Prinzip der Inversen Kinematik als ein Werkzeug zur Bewegung von skelettalen Körpern.

### 2.1 Maschinelles Lernen

Dieser Abschnitt referenziert größtenteils Konzepte aus den beiden Grundlagenwerken *Probabilistic Machine Learning: An introduction* [1] und *Deep Learning* [2]. Angesichts der schnellen Entwicklung von maschinellem Lernen und dem bereits sehr umfangreichen Katalog an Methoden und Praktiken, die darunter fallen, ist es mir kaum möglich hier einen vollständigen Überblick zu geben. Ich habe deshalb einige Details ausgelassen, die zur Einordnung meiner Arbeit in diesen Kontext keinen Mehrwert bieten.

Computer sind seit Jahrzehnten nicht mehr aus der Gesellschaft wegzudenken. Angefangen als automatische Rechenmaschinen hat sich diese Kerntechnologie bis heute extrem weit entwickelt. Und die Entwicklung dauert an und wird kontinuierlich vorangetrieben. In vielen Bereichen hat der Computer den Menschen schnell übertroffen, vor allem wenn es um Zusammenhänge geht, die zwar komplex, aber trotzdem formal durch mathematische Regeln zu beschreiben sind. Schwieriger für den Computer waren allerdings schon immer Dinge, die für uns Menschen intuitiv funktionieren, sich aber schwierig oder gar nicht in einen Satz aus Regeln oder als klares schrittweises Vorgehen beschreiben lassen. Zum Beispiel das Verstehen von Sprache oder das Stellen einer medizinischen Diagnose. Schwierig bedeutet allerdings nicht unmöglich, denn wie man an aktuellen Entwicklungen wie ChatGPT oder Stable Diffusion sehen kann, sind auch diese klassisch menschlichen Aufgaben nicht mehr exklusiv dem Menschen vorbehalten. Wie also schafft man es einen Computer diese scheinbar so menschlichen Fähigkeiten ausführen zu lassen?

Gehen wir einmal davon aus, dass wir einen beliebigen Zusammenhang zwischen einer Eingabe  $x$  und einer Ausgabe  $y$  als mathematische Funktion  $f$  ausdrücken möchten.  $f(x) = y$ . Üblicherweise geht man hier analytisch vor und sucht

ein  $f$ , dass für jedes  $x$  das exakt richtige  $y$  bestimmt. Für klassisch mathematische Zusammenhänge ist dieser Ansatz der bessere Weg, da wir hier eine Funktion  $f$  durch reines Nachdenken und Verstehen des vorliegenden Problems erstellen können. Zum Beispiel lässt sich der Zusammenhang zwischen einem beliebigen Polynom und dessen Ableitung durch eine Funktion beschreiben. Diese Funktion basiert auf wenigen klaren Regeln und hat ein exaktes Ergebnis. Eine perfekte Aufgabe, die ein Computer schneller und effektiver lösen kann als ein Mensch. Diese Herangehensweise hat aber einen entscheidenden Nachteil:

Mit steigender Komplexität des Zusammenhangs steigt auch die Komplexität der zu findenden Funktion rasant an und macht damit die händische Konstruktion dieser Funktion schwierig bis unmöglich. Es ist zum Beispiel intuitiv klar, dass es für den Zusammenhang zwischen Pixelwerten in einem Bild und der Tatsache, ob dort eine Katze zu sehen ist oder nicht, irgendeine Funktion existieren sollte. Schließlich können wir Menschen diese Unterscheidung relativ einfach machen. Die Komplexität dieses Zusammenhangs ist aber so groß und von so vielen Faktoren beeinflusst, dass es nicht möglich ist analog zum analytischen Ansatz eine formale exakte Funktion zu finden. Hinzu kommt, dass sich Zusammenhänge in der realen Welt natürlich ohnehin nie exakt beschreiben lassen. Denn unter Umständen kann das Bild einer Katze auch von Menschen nicht immer als solches identifiziert werden, obwohl es faktisch eines wäre. Mit analytischen Methoden ist dieser Zusammenhang also nicht zu beschreiben, und daher auch auf diesem Weg für einen Computer nicht berechenbar.

Die Alternative ist ein probabilistischer Ansatz. Dazu betrachtet man alle unbekannt Variablen als Zufallsvariablen. Dementsprechend wäre der Zusammenhang zwischen Pixelwerten und dem Inhalt eines Bildes eine Wahrscheinlichkeitsverteilung, die lediglich beschreibt wie wahrscheinlich es ist, dass auf einem Bild ein bestimmter Inhalt zu sehen ist. Das Ziel ist es also nicht länger eine Funktion  $f$  zu finden, die in jedem Fall korrekte Vorhersagen macht. Viel eher geht es darum, sich dieser perfekten Funktion nahe genug anzunähern, so dass sie mit einer gewissen Wahrscheinlichkeit, beziehungsweise Sicherheit, richtige Aussagen macht. Der Prozess solche teilweise korrekten Funktionen zu erstellen und so zu optimieren, dass sie Vorhersagen mit hoher Sicherheit treffen können, ist aber nach wie vor keine einfach zu formalisierende Aufgabe. Maschinelles Lernen beschäftigt sich mit den Methoden und Möglichkeiten genau das zu tun. Tom Mitchell [3] definiert maschinelles Lernen wie folgt:

*A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$ , and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$*

### 2.1.1 Überwachtes Lernen

Was alle Algorithmen des maschinellen Lernens gemeinsam haben ist der Stoff aus dem sie ihre Erfahrung  $E$  überhaupt gewinnen. Denn so ein Algorithmus braucht zuallererst einmal Beispiele aus denen er Muster extrahieren kann. Eine Sammlung aus Beispielen für eine bestimmte Aufgabe  $T$  nennt man häufig einen Datensatz. Unterschieden wird dann, ob ein Algorithmus mit diesen Daten *überwacht* oder *unüberwacht* lernt.

Beim überwachten Lernen sind Beispiele für zu bewältigende Aufgaben bereits vollständig vorgegeben. So gibt es zu jedem Eingabewert  $x$  im Datensatz bereits das



gewünschte Ergebnis der Vorhersage  $y$ , das sogenannte *Label*. Die Eingabewerte setzen sich häufig aus einer Sammlung konkreter Werte für vordefinierte *Merkmale* zusammen. So lernt ein Algorithmus dann, welche wiederkehrenden Merkmale und Merkmalskombinationen von Einzelnen  $x$  auf ein bestimmtes Vorhersageergebnis  $y$  hindeuten.

Ein anschauliches Beispiel für eine Aufgabe, die man mit überwachtem Lernen lösen kann, ist das Erkennen und Zuordnen von handgeschriebenen Ziffern. Ein klassischer Datensatz, den man dafür nutzen kann ist die *MNIST Database of handwritten digits* [4]. Er enthält 70k schwarz-weiß Bilder mit einer Größe von 28x28 Pixel (siehe Abb. 2.1). Jedes Bild zeigt eine einzelne handgeschriebene Ziffer. Die 784 Merkmale für jedes Bild sind die Helligkeitswerte der einzelnen Pixel, die zehn Ziffern des Dezimalsystems ergeben die möglichen Labels. Jedes Bild wurde im Voraus durch einen Menschen mit dem korrekten Label versehen. Solch einen bereits korrekt zugeordneten Datensatz nennt man auch *Ground Truth*.

Eine Aufgabe wie das Bestimmen von Ziffern in einem Bild nennt man auch eine

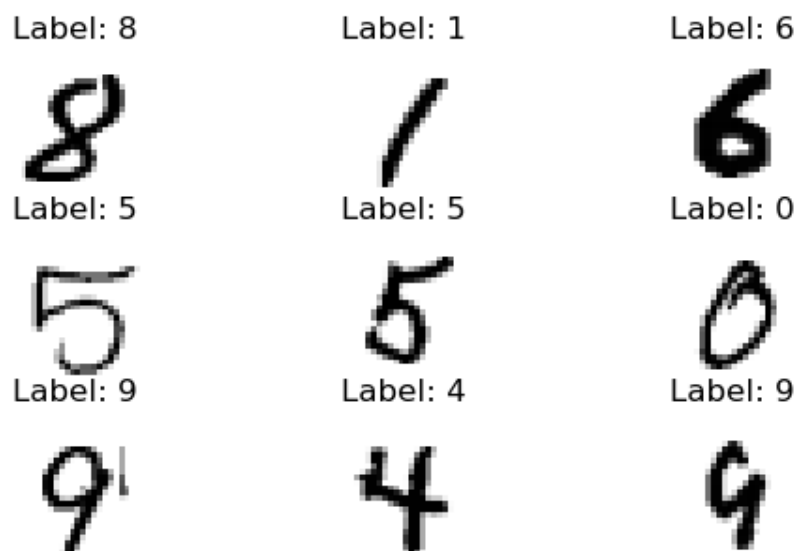


ABBILDUNG 2.1: Visualisierte Beispiele aus dem MNIST Datensatz mit jeweiligem Label.

*Klassifikation*, da der Wertebereich der Vorhersage aus endlich vielen Kategorien besteht. Häufig fallen Aufgaben aber auch in die Kategorie der *Regression*, bei der es stattdessen um die Vorhersage, oder auch das Schätzen eines numerischen Wertes geht. Zum Beispiel ist die Vorhersage von Immobilienpreisen auf Basis von Eigenschaften einer Immobilie eine Regression. Auf diese Aufgabentypen trifft man wohl am häufigsten, es gibt aber auch Aufgaben abseits davon wie zum Beispiel die Übersetzung von Text in eine andere Sprache.

### 2.1.2 Neuronale Netze

Der Bereich des Maschinellen Lernens umfasst sehr viele unterschiedliche Methoden und Praktiken. Ein sehr wichtiger Teilbereich sind die sogenannten *Neuronalen Netze*. Neuronal deshalb, weil sich die Bausteine aus denen sie bestehen grob von den menschlichen Neuronen ableiten lassen. Neuronale Netze erfreuen sich seit den

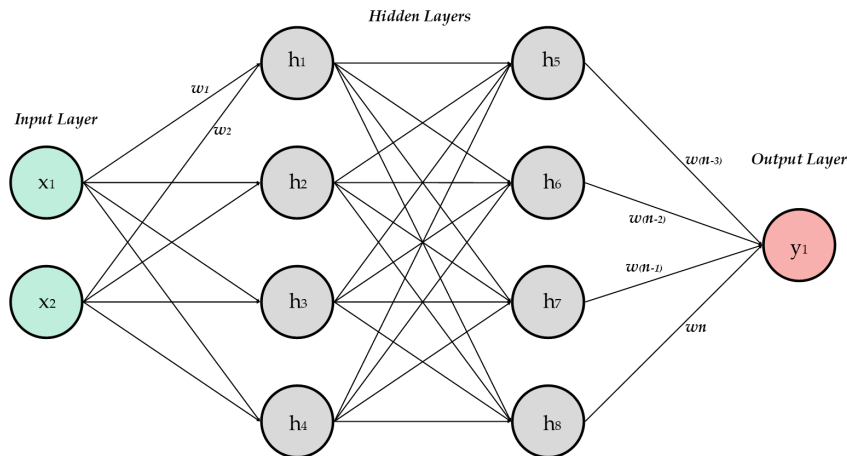


ABBILDUNG 2.2: Ein neuronales Netz mit zwei Hidden Layers

2010er Jahren sehr großer Beliebtheit. Viele der neuen "Künstlichen Intelligenzen" der letzten Jahre sind im Kern Neuronale Netze.

Abbildung 2.2 zeigt ein simples Beispiel für ein solches Netz. Diese Repräsentation als Graph zeigt wie ein Netz im allgemeinen aufgebaut ist. Es gibt eine Eingabeschicht und eine Ausgabeschicht, die über mehrere verborgene Schichten miteinander verbunden sind. Moderne Neuronale Netze bestehen häufig aus einer Vielzahl von Schichten mit sehr vielen Knoten. Aufgrund der dadurch entstehenden *Tiefe* des Netzes spricht man auch von *Deep Learning*.

Jeder Knoten in diesem Graphen stellt eine gewichtete Summe seiner Eingaben dar, gefolgt von einer sogenannten Aktivierungsfunktion  $\varphi(x)$  (siehe Gleichung 2.2). Die Knoten jeder Schicht sind mit den Knoten der Folgeschicht verbunden. Eine einzelne Verbindungen stellt das Gewicht  $w_k$ , mit dem das Resultat des Startknotens in die Summe des Zielknotens einfließt, dar. Die komplexe Funktion  $f$  wird hier durch also durch viele kleine, einfachere Funktionen repräsentiert, die in über mehrere Schichten hinweg miteinander verschachtelt. Die Wahl der Aktivierungsfunktion ist hierbei elementar für die Modellierung komplexer Vorgänge. Nutzt man nur eine lineare Aktivierungsfunktion, kann das gesamte Netz nämlich nur lineare Zusammenhänge darstellen und wäre damit extrem limitiert. Gleichung 2.1 zeigt eine tatsächlich sinnvolle und oft genutzte Aktivierungsfunktion, die *Rectified Linear Unit (ReLU)*. Im Kern schneidet sie lediglich alle Werte ab, die kleiner als Null sind. Das erscheint simpel, trägt aber durch die Nichtlinearität bedeutend zur Funktion des Netzes bei.

$$\varphi(x) = \begin{cases} x, & \text{wenn } x > 0 \\ 0, & \text{andernfalls} \end{cases} \quad (2.1)$$

$$h_1 = \varphi(\mathbf{w}_1 x_1 + \mathbf{w}_2 x_2) \quad (2.2)$$

$$h_5 = \varphi(\mathbf{w}_9 h_1 + \mathbf{w}_{10} h_2 + \mathbf{w}_{11} h_3 + \mathbf{w}_{12} h_4) \quad (2.3)$$

$$y_1 = \varphi(\mathbf{w}_{n-3} h_5 + \mathbf{w}_{n-2} h_6 + \mathbf{w}_{n-1} h_7 + \mathbf{w}_n h_8) \quad (2.4)$$

Die resultierende Funktion aus vielen einfachen, miteinander verbundenen Funktionen kann dann als eine formale Abbildung der Eingabewerte auf einen Ausgabewert beschreiben werden. Gleichung 2.4 zeigt die gewichtete Summe der Ausgabe  $y_1$  aus den Werten der letzten Schicht.  $h_{5-8}$  seien hier ebenfalls wieder Ergebnisse der gewichteten Summen aus  $h_{1-4}$  (siehe Gleichung 2.3). Analog zu Gleichung 2.3 sind diese ihrerseits auch wieder gewichtete Summen der Inputs  $x_1$  und  $x_2$ .

### 2.1.3 Training Neuronaler Netze

Was dieses Konzept zu einem des maschinellen Lernens macht ist der Prozess, wie man sinnvolle Werte für die Gewichte findet. Vor allem, wenn es sich um sehr große Netze handelt, müssen hier mehrere Millionen von Gewichten so aufeinander abgestimmt werden, dass das Netz zu einer sinnvollen Näherung von  $f$  wird. Dazu werden Neuronale Netze mit einem Datensatz *trainiert*. Das Training besteht aus einem klassischen iterativen Optimierungsprozess. Alle Gewichte werden mit zufälligen Werten initialisiert und über viele Wiederholungen immer wieder angepasst, bis die Vorhersagen des Netzes ausreichend nah an den Labels im Datensatz liegen. Dafür hängt man an das Netz zuerst noch eine zusätzliche Fehlerfunktion an, die die Ausgabe des Netzes mit der Ground Truth vergleicht. Die Wahl dieser Fehlerfunktion ist relevant für den Trainingserfolg und die einzige Voraussetzung ist, dass sie, genau so wie alle Aktivierungsfunktionen, differenzierbar sein muss. Geht es zum Beispiel um die Regression von Immobilienpreisen, so könnte eine einfache Fehlerfunktion die quadrierte Differenz zwischen Ground Truth und der Schätzung des Netzes sein. Da alle Teile eines Netzes ableitbar gestaltet sein sollten, ist es möglich den Gradienten der Fehlerfunktion nach einem beliebigen individuellen Gewicht im Netz zu berechnen. Die ineinander verschachtelten Einzelfunktionen des Netzes können mithilfe der Kettenregel „ausgerollt“ und differenziert werden. Da man bei diesem Vorgehen mit dem Differenzieren der tiefsten Knoten beginnt und sich rückwärts durch das Netz zum Zielknoten vorarbeitet, nennt man diesen Vorgang auch *Backpropagation*. Statt die Eingabewerte von vorne nach hinten durch zu reichen, wird hier das Ergebnis der Fehlerfunktion rückwärts durch den Graphen gereicht. In einem zweiten Schritt werden mithilfe eines Optimierers die Gewichte in Richtung eines kleineren Fehlers angepasst. Diese beiden Schritte werden für Einzelne oder eine Gruppe von Eingabe/Ausgabe Paaren aus dem Datensatz solange wiederholt, bis der Fehler sich einem globalen Minimum genähert hat. Die Wahl eines angemessenen Optimierers ist ähnlich wie die Wahl einer Aktivierungsfunktion und der Fehlerfunktion ein relevanter Schritt in der Konzeption eines Netzes.

Ein möglichst kleiner Fehlerwert alleine ist aber noch kein eindeutiges Indiz für ein funktionierendes Netz. Da man das Netz nur auf einer Stichprobe trainiert, muss die Fähigkeit des Netzes auf ungesehenen Daten zu generalisieren, getestet werden. Man kann für diesen Zweck zum Beispiel einen kleinen Teil des Datensatzes von vorneherein vom Training ausschließen und zur Evaluation der Erkennungsfähigkeit nutzen. Daten zur reinen Evaluation eines Netzes nennen sich *Testdaten*. Zur Veranschaulichung der Notwendigkeit von Testdaten zur Bewertung eines Netzes stelle man sich vor, das Netz erreicht im Training einen Fehler von Null auf allen Trainingsdaten. Entgegen der ersten Annahme spricht das üblicherweise für ein potenziell schlechteres Ergebnis. Ein niedriger Fehler bedeutet, dass das Netz perfekte Vorhersagen auf den Trainingsdaten macht, was höchstwahrscheinlich bedeutet, dass es lediglich den Trainingsdatensatz auswendig gelernt hat statt generalisierbare Muster zu finden. Dieses Phänomen nennt man *Overfitting*. Erreicht man einen

annehmbaren Fehler beim Training, der sich mit den Testdaten reproduzieren lässt, so kann man davon ausgehen, dass das Netz tatsächlich relevante Muster in den Daten gelernt hat.

### 2.1.4 Convolutional Neural Networks

Für kleinformatige Bilder, wie die im MNIST Datensatz, sind klassische Neuronale Netze wie zuvor besprochen eine akzeptable Lösung. Versucht man allerdings größere, farbige Bilder mit entsprechend komplexerem Inhalt als Eingabe zu verwenden stößt man vor allem auf drei zentrale Probleme.

1. Die klassischen Netze arbeiten mit einer festen Größe des Eingabevektors. Ändert sich dieser, ändert sich damit auch die Menge der zu lernenden Gewichte. Bilder gibt es aber in den unterschiedlichsten Formaten während der Inhalt eines Bildes völlig unabhängig vom Format ist. Lernen wir also eine Aufgabe, die etwas auf einem Bild erkennen soll, ist es unsinnig sich auf ein festes Bildformat festzulegen, da der gleiche Inhalt auch auf einem andersformatigen Bild zu sehen sein kann und dabei die gleichen Merkmale aufweist.
2. Bilder sind translationsinvariant. Das bedeutet Objekte in einem Bild können an beliebigen Stellen im Bild auftauchen und dabei die selben Merkmale entfalten. Ein Netz, das für jeden Pixel ein einzelnes Gewicht hat, kann vielleicht eine Orange in der linken oberen Ecke erkennen aber nicht in der unteren rechten Ecke. Die Gewichte für die Pixel in diesen beiden Bereichen sind schlichtweg unterschiedlich und unabhängig voneinander.
3. Für etwas größere Bilder, zum Beispiel Bilder mit drei Farbkanälen und 512x512 Pixeln, hätten wir allein in der Eingabeschicht fast 800k Knoten. Dadurch explodiert bereits für eine geringe Anzahl Neuronen in den versteckten Schichten die Anzahl der Gewichte. Bei brauchbar vielen Schichten und Knoten ist die dadurch entstehende Menge an Gewichten auch für heutige Computer nicht handhabbar.

Zur Lösung dieser Probleme, kann man sich zunutze machen, dass Pixel die nah beieinander liegen, räumlich zusammen hängen. Beim Finden von relevanten Mustern können wir also annehmen, dass diese nicht aus dem Zusammenspiel aus den 4 Eckpixeln des Bildes bestehen, sondern viel eher aus z.B. 3x3 Pixel großen Patches. Nimmt man die Translationsinvarianz hinzu, lässt sich festhalten, dass diese Merkmale an beliebigen Stellen im Bild zu finden sein können.

Netze, die solche räumlichen Muster lernen, nennen sich *Convolutional Neural Networks* (CNNs), weil sie die mathematische Operation der Konvolution nutzen um diese Muster zu kodieren. Abbildung 2.3 zeigt einen sogenannten *Filter*, den Kern der Konvolutionsoperation. Die Operation besteht nun darin, diesen Filter als Maske auf jeden Pixel des Bildes zu legen und entsprechend der Faktoren im Filter, die Farbwerte der Nachbarpixel und des Ursprungspixels (gelb unterlegt) auf zu summieren. Der Filter kann auch größer gewählt werden um mehr Nachbarpixel in die Berechnung mit einzubeziehen. So erhält man für jeden Pixel im Bild einen neuen Wert, der Informationen aus dessen Umfeld enthält. Ein spezieller Anwendungsfall wäre zum Beispiel das Erzeugen künstlicher Unschärfe, indem man den Wert des Originalpixel mit dem Mittelwert aus allen umgebenden Pixeln ersetzt. Filter können aber natürlich auch wie in Abbildung 2.3 Werte enthalten, die sich nicht wieder zu 1 aufsummieren lassen. Trotzdem hat das Ergebnis so einer Konvolution

eine Bedeutung, die sich jetzt nur nicht mehr trivial visualisieren lässt. Abbildung 2.4 zeigt das Ursprungsbild und eine Visualisierung der pro Pixel Konvolution mit dem Filter aus Abbildung 2.3 (Für diesen Zweck wurde aus den RGB Daten des Farbbildes zuvor die Helligkeit pro Pixel errechnet). Für alle Pixel, bei denen zwischen den Nachbarpixeln links und rechts kaum ein Helligkeitsunterschied besteht ergibt die Konvolution einen Wert nahe 0. Gibt es allerdings einen Helligkeitsunterschied spiegelt sich das im Betrag des Konvolutionsergebnisses wieder. Wie man am Ergebnisbild erkennen kann, berechnet dieser spezielle Filter, ob in der Umgebung eines Pixels eine harte vertikale Kante zu erkennen ist.

1	0	-1
2	0	-2
1	0	-1

ABBILDUNG 2.3: Alle Kantenpixel bei denen es von links nach rechts heller wird, haben einen niedrigen Wert. Ein Helligkeitsunterschied von rechts nach links ergibt einen hohen Wert.

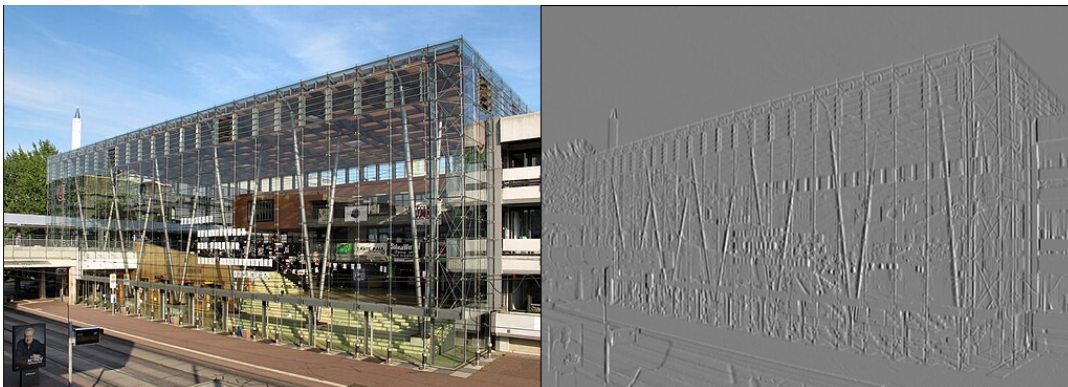


ABBILDUNG 2.4: Jeder Pixel in dem rechten Bild, beschreibt ob im unmittelbaren Umfeld seines Originals eine vertikalen Kante ist. Grau bedeutet 0 als Ergebnis des Filters, schwarz einen niedrigen Wert und weiß einen hohen. Quelle: Blutgretchen, CC BY-SA 3.0 <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons

Die Konvolution kann also für jeden Pixel, bzw. Ort im Bild, berechnen, ob dort ein relevantes Merkmal vorhanden ist. Was genau dieses Feature ausmacht bestimmen die Beträge und die Anordnung der Faktoren im Filter. Es bietet sich also an die Faktoren eines Kerns als Merkmalsrepräsentation für Bilder zu nutzen. Da ein Netz das Erkennen von Merkmalen lernen soll, liegt es nahe die Faktoren eines Kerns als lernbare Gewichte zu nutzen. Ein einzelner 3x3 Filter mit 9 Gewichten würde dann aufgrund der Eigenschaften von Bildern ein einzelnes Merkmal eines einzelnen Kanals des gesamten Bildes abdecken. Bilder können viele verschiedene



Merkmale aufweisen, weshalb CNNs mit vielen verschiedenen Filtern pro Schicht arbeiten. Die Menge der Filter pro Schicht nennt man die Kanäle dieser Schicht, analog zu den drei Farbkanälen in der Eingabe. Abbildung 2.5 zeigt eine schematische Darstellung des Konvolutionsschrittes in einer einzelnen Schicht mit 3 Eingabe und 2 Ausgabe Kanälen.

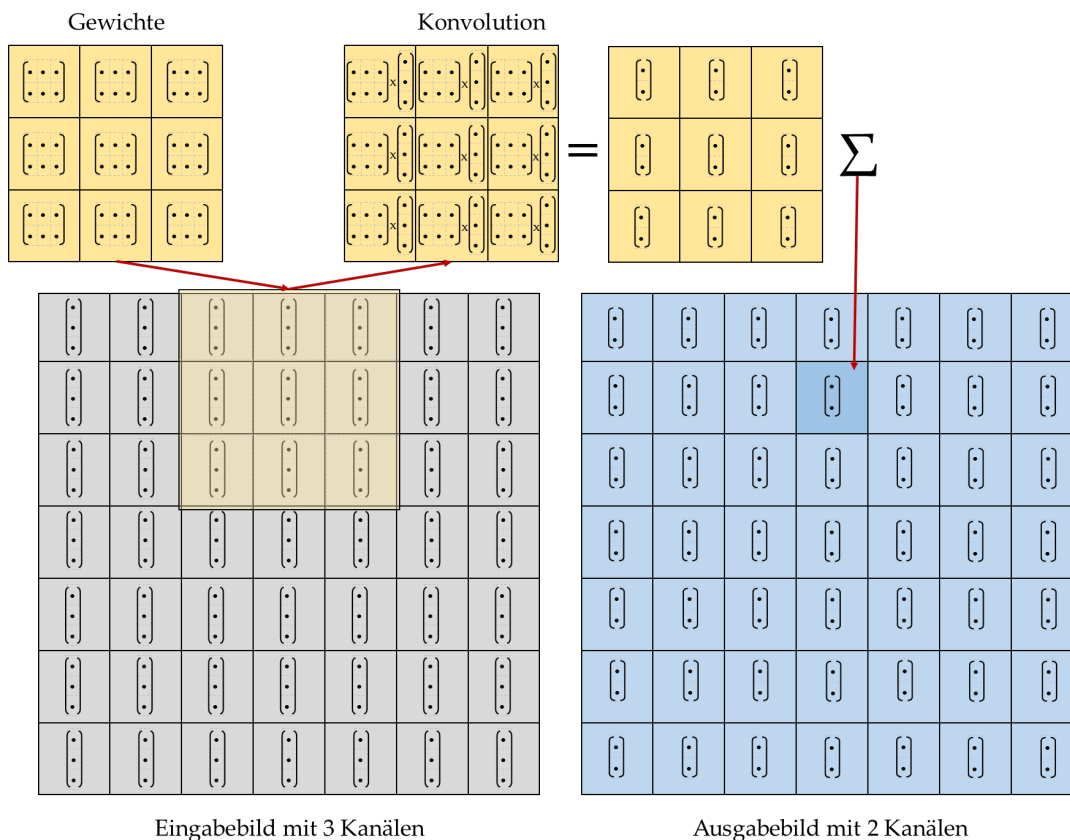


ABBILDUNG 2.5: Eine schematische Darstellung von mehrkanaliger Konvolution in CNNs. Adaptiert von den Vorlesungsfolien aus dem Kurs *Deep Learning und 3D Bildverarbeitung* von Udo Freses.

Ein Problem dieses Ansatzes ist bisher, dass ein 3x3 Filter keine Merkmale erkennen kann, die sich erst in Patches von 30x30 Pixeln erkennen lassen würden. Das Vergrößern der Filter würde zu einer quadratisch wachsenden Menge Gewichte führen und man hätte für sehr große Filter dieselben Probleme wie klassische Neuronale Netze. Man bleibt daher bei kleinen Filtern, verringert aber über tiefer werdende Schichten schrittweise die Auflösung des Bildes. Dadurch gehen zwar Details verloren, die Theorie ist aber, dass diese kleineren Details in vorigen Schichten bereits erkannt werden und die größeren Details trotz der Herunterskalierung erhalten bleiben. Diese Herunterskalierung nennt man *Pooling* und sie besteht oft einfach nur aus einer Zusammenfassung von vier beieinanderliegenden Pixeln zu einem einzigen. Hier unterscheidet man zwischen Max-Pooling (das Maximum der vier Werte ist der neue Wert) und Average-Pooling (Der Durchschnitt der vier Werte ist der neue Wert).

Darüber hinaus greift dann wieder das gleiche Prinzip wie bei klassischen Neuronalen Netzen, bei dem Merkmale aus dem Eingabebild immer weiter über immer

tiefere Schichten mit anderen Merkmalen zu neuen Merkmalen verschachtelt werden. Carter et al [5] zeigen dazu sehr schön, wie ein Netz zur Bildklassifizierung über seine Schichten hinweg eine Transformation von bildhaften Merkmalen wie Kanten, Farben und einfachen Formen über komplexere Muster und Farbkombinationen hin zu semantischen Features wie „Arm“, „Frucht“ oder „Wasser“ lernt.

## 2.2 Trainingsdaten

Die Beschaffung von Trainingsdaten für alle Anwendungen des maschinellen Lernens ist keine triviale Aufgabe, denn meist muss die zu lernende Aufgabe von Menschen im Voraus selbst erledigt werden. Das Ziel ist es aber gerade, diese Aufgabe zu automatisieren. Da es in meiner Arbeit grundsätzlich um das Training von CNNs für die Verarbeitung von Bildern geht, werde ich mich hier hauptsächlich mit den Details zu bildhaften Trainingsdaten und deren Beschaffung und Erzeugung beschäftigen, wobei ich damit nicht ausschließe, dass die Prinzipien auch für andere Trainingsdaten gelten.

Das Bewältigen der zu lernenden Aufgabe für die Daten eines Bilddatensatzes zwecks überwachtem Lernen nennt man *Annotation*. Die im Voraus erforderliche Beschaffung von ausreichend viel angemessenem Bildmaterial ist allerdings auch eine elementar wichtige Aufgabe. Ein großer bestehender Datensatz ist der ImageNet Datensatz für die Klassifikation von Bildern [6]. Er bestand zum Veröffentlichungszeitpunkt aus 3.2 Millionen Bildern, die zu 99.7% korrekt in 5247 inhaltliche Kategorien eingeordnet, also annotiert worden sind. Das Ziel war es für all diese Kategorien 500-1000 Beispielbilder zu finden. Dafür mussten die Autoren nicht nur eine repräsentative Menge unterschiedlicher Kategorien festlegen, sondern auch das Internet nach vielen Millionen von Kandidatenbildern absuchen, die daraufhin dann von Menschen gefiltert und annotiert werden mussten.

Soll ein Netz komplexere aber verwandte Aufgaben wie zum Beispiel neben der reinen Klassifizierung des Bildinhaltes auch die Lokalisierung des erkannten Objektes im Bild lernen, ist eine zusätzliche Annotation erforderlich. Für die *ImageNet Large Scale Visual Recognition Challenge* [7] konstruierten Russakovsky et al. drei unterschiedliche Datensätze auf Basis des ImageNet Datensatzes um Netze für drei verschiedene Aufgaben der Bildverarbeitung zu testen.

Wichtig zu verstehen ist, dass ein gelernter Algorithmus immer nur so gut ist, wie seine Trainingsdaten. Ein Algorithmus zur Vorhersage von Immobilienpreisen könnte sehr genaue Vorhersagen für Immobilien im mittleren Preissegment machen, weil es dafür im Datensatz viele Beispiele gibt. Hier kann der Algorithmus seine Erfahrung anwenden um effektiv zwischen den Beispielen zu interpolieren. In den selteneren Preissegmenten wird diese Interpolation etwas ungenauer, da bei wenigen Beispieldaten das natürliche Rauschen dieser Daten zunehmend größeren Einfluss hat. Noch schwieriger wird es für den Algorithmus, wenn er eine Vorhersage für eine Immobilie treffen soll, deren Preis außerhalb der Beispiele im Datensatz liegt. Hier muss das Netz versuchen die Muster aus den Beispieldaten zu extrapolieren, statt dazwischen zu interpolieren. Das ist gleichzeitig eine Schwäche als auch eine Stärke von Methoden des maschinellen Lernens. Logischerweise ist die Sicherheit, mit der die resultierende Vorhersage korrekt ist, geringer. Die Tatsache, dass es überhaupt komplexe Muster extrapolieren kann ist aber beeindruckend. Ein CNN

Klassifizierer hätte beispielsweise Schwierigkeiten einen Dalmatiner korrekt zu erkennen, wenn es im Training nur Beispiele für Golden Retriever und Dackel gesehen hat. Es ist aber gut möglich, dass das Netz trotzdem das grundlegende Konzept "Hund" gelernt hat und dadurch zumindest eine sinnvolle Vorhersage machen kann.

Möchte man sich allerdings nicht auf die Extrapolationsfähigkeiten eines Netzes verlassen, bedarf es aufgabenspezifischer Trainingsdaten. Das bedeutet oft, dass man sich bei der Suche nach Datensätzen nicht auf vorhandene Datensätze oder Bilder aus dem Internet verlassen kann. Ein Ernteroboter, der mithilfe eines CNNs bestimmte Früchte auf einem Feld erkennen soll, bräuchte zum Beispiel einen konkret auf den Anwendungszweck zugeschnittenen Datensatz um die besten Ergebnisse zu erzielen. Sowohl die Daten selbst als auch die Annotation der Daten müssen dann für genau diese Aufgabe erst erzeugt werden. Insbesondere für die Annotation realer Bilder ist man da abseits von einigen Tricks oder komplizierten Laborumgebungen auf Menschen angewiesen.

Ein Problem der Annotation durch Menschen entsteht vor allem dann, wenn sich die gewünschte Ausgabe des Netzes für Menschen schlecht formalisieren lässt oder extrem aufwendig ist. In Abbildung 2.6 ist zum Beispiel die Segmentierung von einzelnen Kuh-Instanzen zu sehen. Solch eine Segmentierung ist für den Menschen im Kopf trivial, lässt sich aber nur mit verhältnismäßig großem Aufwand pixelgenau in eine Form bringen, mit der das Netz arbeiten kann. Auch die Posenbestimmung von Objekten ist eine Aufgabe, bei der die Annotation deutlich schwieriger ist als eine einfache Zuordnung zwischen Wörtern und Bildern oder der Lokalisation einer Feldfrucht im Bild [8].



ABBILDUNG 2.6: Hier links zu sehen das Originalbild von Kühen und rechts die pro Pixel Ground Truth einer Segmentierung von Kuh-Instanzen, die ich von Hand in einer Bildbearbeitungssoftware annotiert habe. *Quelle des Originals: Markus Stenzel auf pixabay <https://pixabay.com/photos/cows-nature-cattle-bovine-country-61659/>*

### 2.2.1 Synthetische Trainingsdaten

Ein hilfreiches Werkzeug, mit dem man diesem Aufwand entgegenwirken kann ist, einen Teil der Daten durch synthetische Daten zu ersetzen oder sogar ganz auf reale Daten zu verzichten. Synthetische Daten meint, dass für das Training relevante



Szenen, Bilder oder Objekte am Computer simuliert werden. Großer Vorteil dieser Herangehensweise ist, dass der Prozess der Erzeugung der Bilder auch gleich ohne oder mit wenig extra Aufwand die Annotation mitliefert. Außerdem ist es mithilfe einer Simulation bedeutend einfacher große Mengen von Bildern zu erzeugen oder bestimmte Überparameter im Datensatz anzupassen und die Ergebnisse zu beobachten. Letztendlich können simulierte Daten allerdings die Realität noch nicht zu 100% abbilden. Es ist deshalb wünschenswert diese *Reality Gap* zu überbrücken um synthetische Daten effektiv nutzen zu können. Dabei ist die exakte Nachbildung von Szenen von Realdaten nicht nur zu aufwendig, es liefert auch weniger gute Ergebnisse. Viel besser funktionieren stark randomisierte Umgebungen, da das dem Netz erlaubt sich besser auf relevante Merkmale zu konzentrieren. Dadurch erzielen rein synthetisch trainierte Netze sehr gute Ergebnisse, die sich durch zusätzliche Verfeinerung auf realen Daten sogar noch verbessern lassen [8], [9].

Tobin et al. [9] sprechen konkret vom Konzept der *Domain Randomization*. Hier legt man entsprechend wenig Wert auf einen realistischen Aufbau der simulierten Szenen und versucht stattdessen maximal diverse Daten zu erzeugen. Die Autoren randomisieren Parameter wie Position, Textur und Anzahl der Objekte sowie den Hintergrund, die Belichtung und auch die Kameraparameter und erzeugen einen vollständig synthetischen Bilddatensatz ohne fotorealistic Render Engine. Ein ausschließlich darauf trainierter Greifroboter konnte das in der Simulation gelernte auch in der Realität sinnvoll anwenden. Die zu erkennenden Objekte waren hier allerdings auch nur simple geometrische Objekte.

Bousmalis et al. [10] untersuchen den Einfluss von simulierten Trainingsbildern für robotisches Greifen und beobachten eine deutliche Verbesserung der Erfolgsrate für Griffe, wenn zusätzlich zu realen Daten mit Daten aus simulierten Szenen trainiert wurde. Außerdem konnten sie diese Ergebnisse noch weiter verbessern, indem sie die simulierten Szenen zusätzlich randomisiert haben. Die simulierten Szenen sind hier allerdings auch virtuelle Replika des echten Szenen Aufbaus.

Rudorfer et al. [11] kommen ebenso zu dem Schluss, dass vor allem Hintergrundvarianz und die Zusammenstellung der Szenen einen hohen Einfluss auf die Erkennungsfähigkeiten von Netzen haben.

Die intuitive und wohl möglich effektivste Methode die Reality Gap zu überbrücken ist allerdings das Rendern fotorealistischer Bilder. Die Szene mag randomisiert sein, die visuelle Qualität sollte der von echten Bildern allerdings möglichst nahe kommen. Dazu gibt es unterschiedliche Ansätze. Mueller et al und Bousmalis et al [10], [12] re-stilisieren synthetischen Daten mit einem GAN zu realistisch wirkenden Bildern für das Training eines Handposen-Regressors und eines Greifroboters und berichten von einer Verbesserung ihrer Ergebnisse durch diese Methode. Anzumerken ist aber, dass Mueller et al [12] mit einer simplen Augmentation der Daten durch zufällige Gamma Korrektur ebenfalls einen ähnlich großen Leistungssprung beobachten konnten.

Elementar für realistische Renderings simulierter Szenen sind allerdings vor allem die Methoden des physikbasierten Renderings *PBR*. Kern dieser Methoden ist die realistische Simulation der Interaktion zwischen Licht und Materie mithilfe von Funktionen, basierend auf physikalischen Gesetzen [13]. Unterstützt ein Renderer *PBR*, so erlaubt er dem Anwender meist eine feine Parametrisierung von Materialeigenschaften eines Objektes. Abbildung 2.7 zeigt wie durch verschiedene Parameter wie Rauheit, Metallische Reflexion und Transparenz verschiedenste Materialien wie Plastik, Glas und Metalle simuliert werden können.

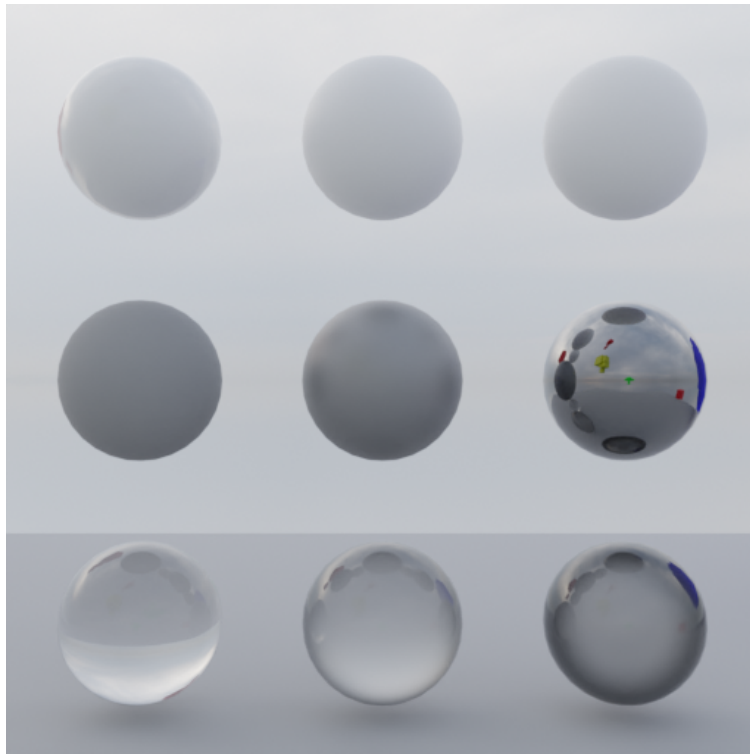


ABBILDUNG 2.7: Oben ein Material mit abnehmender Rauheit. In der Mitte ein Material mit abnehmender Rauheit und zunehmender metallischer Reflexion. Unten ein Glas Material mit zunehmender innerer Rauheit.

### 2.3 6D Posenbestimmung von Objekten

Die Erkennung von Objekten und die Bestimmung von deren Pose aus Bilddaten ist eine wichtige Aufgabe für diverse Bereiche der Computer Vision wie autonomes Fahren, Robotik und Augmented Reality. In der Computergrafik hat die Pose eines statischen Objektes 6 Freiheitsgrade. Drei für die Translation entlang der  $x, y$  und  $z$  Achsen sowie drei für die Rotation um die drei lokalen Achsen des Objektes.

Herkömmliche, nicht lernbasierte Methoden, schätzen die Objektpose indem sie zwei- oder dreidimensionale Merkmale von Objekten mit einem Datensatz aus Beispielposen abgleichen und die passendste Pose finden. Die regelbasierte Extraktion von zweidimensionalen Merkmalen aus Bildern ist allerdings wenig robust gegenüber kleinen Unterschieden im Aufbau, Texturierung und der Beleuchtung der Szene. Dreidimensionale Merkmale sind zwar aussagekräftiger aber auch schwieriger zu extrahieren, da man RGB-D Bilder von Tiefenkameras oder Scans von 3D Scannern benötigt [14].

Deep Learning Ansätze mit CNNs können 6D Objektposen mittlerweile ziemlich genau aus reinen RGB Bildern und auch aus RGB-D Bildern schätzen und haben damit klassische Methoden der Posenbestimmung in Sachen Genauigkeit abgelöst, wie die Ergebnisse des aktuellsten *Benchmark for 6D Object Pose Estimation* (BOP) [15] zeigen. Methoden, die vollständige 6D Posen erkennen arbeiten häufig nur auf der Instanz Ebene. Das bedeutet sie werden konkret auf einer Sammlung interessanter Objektinstanzen trainiert, deren 3D Modelle bekannt sind, und generalisieren dafür sehr

schlecht auf anderen Objekten, wenn überhaupt. Methoden auf der Kategorie Ebene können häufig nur 3D Objektposen und deren Ausmaß als 3D Bounding Box bestimmen, funktionieren indes aber für gewöhnlich auch für ungesehene Objekte antrainierter Objektkategorien. [16].

Eines der frühesten CNNs für diese Aufgabe der 6D Posenbestimmung ist *PoseCNN* von Xiang et al. [17]. *PoseCNN* ist auf bestimmten Objektinstanzen trainiert und lernt die Koordinatentransformation vom Objektkoordinatensystem zum Kamerakoordinatensystem eines gezeigten Bildes. Dabei verfolgen die Autoren den Ansatz die Erkennung der Rotation von der Erkennung der Translation innerhalb des Netzes zu entkoppeln, da beide Parameter separate Einflüsse auf die Erscheinung des Objektes im Bild haben. Die Translation beeinflusst die Größe und Position im Bild. Die Rotation dagegen beeinflusst die Form und die Textur des Objektes im Bild.

Trainiert wurde *PoseCNN* auf knapp 130.000 realen RGB-D Bildern extrahiert aus 80 eigens aufgenommenen und annotierten Videos, die um 80.000 synthetische Bilder ergänzt wurden. Die genutzten Objektinstanzen sind eine Teilmenge aus dem YCB-Datensatz [18] für robotische Manipulationsaufgaben. Dieser Datensatz beinhaltet eine Menge von realen Objekten und dazugehörigen detaillierten 3D Scans dieser Objekte. Die Ergebnisse von *PoseCNN* waren sehr vielversprechend und zeigten wie mächtig CNNs auch in diesem Bereich sein können.

Bei der lernbasierten 6D Posenbestimmung von Objekten gibt es aktuell kein Netz, das ausschließlich auf realen Daten trainiert wurde. Synthetische Daten sind für diesen Bereich also fast unerlässlich. Das liegt unter anderem daran, dass die Annotation von realen Daten eben so aufwendig ist, dass reale Datensätze schlichtweg nicht groß genug sind. Beim BOP-Benchmark 2020 [19] bestanden fast alle bereitgestellten Datensätze aus fotorealistischen Renderings. Als Testdaten werden dabei nach wie vor ausschließlich reale Bilder verwendet. Auch in der BOP-Challenge von 2022 [15] sind weiterhin alle teilnehmenden Neuronalen Netze mindestens teilweise auf synthetischen Daten trainiert. Den zweiten Platz hält ein Netz das vollständig auf synthetischen RGB Bildern trainiert wurde. Nur dasselbe Netz mit ergänzenden realen Daten konnte noch bessere Ergebnisse erzielen.

### 2.3.1 Hand-Objekt Datensätze

Methoden, die die Objektpose bei konkreten Hand-Objekt Interaktion bestimmen sind allerdings verhältnismäßig unerforscht. Chao et al. [20] stellen aber zum Beispiel fest, dass die Erkennungsfähigkeit eines klassischen Netzes deutlich sinkt, wenn es gegriffene Objekte erkennen muss. Ob dieser Effekt lediglich auf die dadurch entstehende Verdeckung zurückzuführen ist bleibt offen.

Datensätze, die annotierte Hand-Objekt Interaktionen enthalten, gibt es jedoch deutlich mehr. Das liegt vor allem daran, dass die meisten Datensätze ebenfalls die Pose der Hand annotieren, was für viele Anwendungen, die eine Hand-Objekt Interaktion simulieren wollen, essenziell ist.

Brahmblatt et al. [21] sammeln annotierte Realbilder von gegriffenen Objekten und rekonstruieren die Handposen aus RGB-D Bildern. Die gegriffenen Objekte dafür sind allerdings 3D gedruckt und mit Motion Capture Markern versehen. Leibe et al. [22] konstruieren einen vollständig per Pixel annotierten Datensatz gegriffener Objekte. Dabei enthält die Ground Truth die Objektpose sowie die Position der Fingerspitzen. Dieser Datensatz enthält jedoch nur eine sehr limitierte Auswahl einfacher Objekte und ist mit 3k Bildern nur sehr klein. Cao et al. [23] annotieren Realbilder

Name	Art	Auflösung	# Bilder	# Objekte	Annotation
Cao et al [23]	real	-	500	125c	g/m
HONotate [26]	real	640×480	78K	10	a
DexYCB [20]	real	640×480	582K	20	m
Hasson et al[27]	synt	256×256	154k	3k	g
GANHand [28]	synt/real	-	130k	58	g
FPHA [25]	real	1920×1080	105K	4	s
ContactPose [21]	real	960×540	2,991K	25	M
Dexter+Obj [22]	real	640×480	3K	2	m
ARCTIC [24]	real	2560x1440	2,100k	11	M

TABELLE 2.1: Eine Übersicht aller genannter Datensätze. Der Datensatz von Cao et al [23] beschreibt Objektkategorien und nicht Instanzen, weshalb er verhältnismäßig viele verschiedene Objekte enthält. *g*: generiert, *a*: automatisch, *m*: manuell, *M*: Motion Capture, *s*: sensorisch

durch einen mehrschrittigen Prozess aus lernbasierter Rekonstruktion des Handmeshes, Schätzung der Objektpose und mehreren Optimierungsschritten zur Zusammenführung von Hand und Objekt sowie der Verfeinerung der Handpose um realistischere Kontaktflächen zu erhalten. Dabei erforderte die Rekonstruktion der Hand ein manuelles Refinement. Für diesen Datensatz wurden Objekte in Bildern aber nur durch ähnliche Objekte für die die 3D Modelle vorhanden waren approximiert. Fan et al. [24] entwickelten einen Hand-Objekt Datensatz aus realen Daten inklusive beidhändiger Manipulationsaktionen mithilfe von Motion Capture aus mehreren Kamerawinkeln. Sowohl auf Händen als auch auf Objekten sind daher aber Motion Capture Marker sichtbar, wenn auch unscheinbar. Garcia-Hernando et al. [25] erzeugen einen Datensatz aus RGB-D Videodaten und sensorisch annotierter Handpose. Die Objektpose ist dabei allerdings nur in einem Teil der aufgenommenen Videos enthalten. Hampali et al. [26] konstruieren einen Datensatz aus den Bildern mehrerer synchronisierter RGB-D Kameras und erarbeiten aus diesen Daten eine vollautomatische Annotation der Bilder. Hasson et al. [27] bringen einen vollständig synthetischen Datensatz vor, für den die simulierten Griffposen automatisch mithilfe eines Algorithmus aus der Robotik erzeugt wurden. Dieser Algorithmus ist allerdings in seiner Posenvielfalt und Plausibilität eingeschränkt. Corona et al. [28] produzieren ebenfalls einen Datensatz aus gegriffenen Objekten aus dem YCB Datensatz und verwenden dafür gerenderte Hände auf realem Hintergrund, erstellen die Griffe jedoch zwecks Posenvielfalt händisch.

Chao et al. [20] konstruieren mit DexYCB den meines Erachtens nach umfangreichsten Datensatz von von Hand-Objekt Interaktionen aus realen Videoaufnahmen und annotieren diesen vollständig manuell von Hand mithilfe von kontextuellen Keypoints für Hand und Objekt. Sie nutzen dafür Objekte aus dem beliebten YCB-Set [18], das oft auch zur Objektposenbestimmung genutzt wird.

Fast alle der hier genannten Datensätze enthalten ausschließlich oder in Teilen reale Daten und digitalisieren Handposen indem sie reale Hände aufnehmen und die Handpose anhand von Markern oder Tiefendaten schätzen. Der einzige vollständig synthetische Datensatz verwendet einen kategorisierten Objektdatensatz mit sehr vielen unterschiedlichen Instanzen und rendert zudem keine vollständigen Szenen sondern nur gegriffene Objekte auf Foto Hintergründen. Ich konnte keinen synthetischen Datensatz aus vollständig simulierten Szenen mit Hand-Objekt Interaktionen finden.

## 2.4 Handmodelle

Hände sind ein elementarer Bestandteil des menschlichen Körpers, werden aufgrund ihrer Komplexität aber im Virtuellen häufig vereinfacht dargestellt. Der Mensch benötigt diese Komplexität im echten Leben um mit seiner Umwelt im Großen wie auch im Kleinen zu interagieren. Gleichmaßen ist der Mensch trotz der Komplexität sehr gut darin Handbewegungen zu interpretieren und unnatürliche Handbewegungen zu erkennen [29]. Wenn Plausibilität und Akkuratheit der darzustellenden Hand auch im virtuellen eine Rolle spielen, muss diese Komplexität in die gewünschten virtuellen Umgebungen übertragen werden. Das gilt zum Beispiel für Anwendungen der Human Computer Interaction für VR und AR [30], Übersetzer für Zeichensprache, Biometrische Scanner oder soziale Roboter [31].

Um die Bewegung und das Aussehen von Händen in eine virtuelle Umgebung abzubilden wird ein formales Modell der Hand benötigt, das reale Handposen akkurat modellieren kann. Für Anwendungen die nicht nur an Posendaten interessiert sind, sondern auch an einer visuell akkuraten Modellierung der Hand, braucht es zusätzlich zu einem skelettalen Model auch ein Modell der Handoberfläche, möglicherweise auch inklusive realistischer Texturen. Ziele dieser Handmodelle sind oft die Repräsentation einer Handpose über eine gewisse Menge numerischer Parameter, weshalb man sie auch als *parametrische* Handmodelle bezeichnet.

Zhao et al [32] nutzen ein Modell der Hand mit 33 Freiheitsgraden zur Anwendung der Handposenrekonstruktion aus Motion Capture Daten. In Abbildung 2.8 ist zu sehen wie die Bewegungsfreiheit der Gelenke in Richtung des Handgelenks zunimmt. Auch zu sehen ist, dass der Daumen komplexere Bewegungen ausführen kann als die restlichen vier Finger, da er insgesamt 7 Freiheitsgrade hat statt nur 5. Auch [30] sprechen von der Schwierigkeit die Ausdrucksfähigkeit des Daumens zu modellieren und beziehen sich dabei zum Beispiel auf eine biomechanische Studie, die die Rotationsachsen des ersten Daumengelenks untersucht hat und feststellt, dass dieses Gelenk zwei voneinander unabhängige Rotationsachsen hat [33].

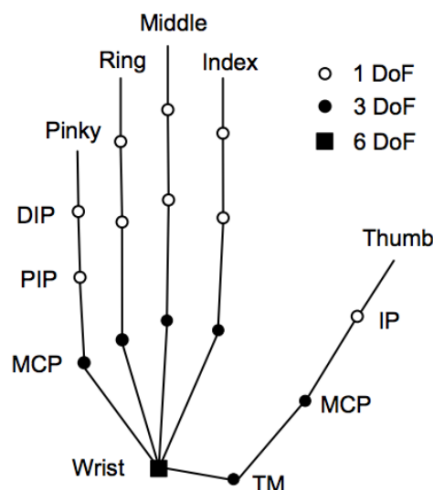


ABBILDUNG 2.8: Die Repräsentation der Hand genutzt von [32] für die Rekonstruktion von Motion Capture Daten



Eine visuell realistische Repräsentation mit einer Hautoberfläche hat neben dem Erscheinungsbild den Vorteil, dass auch der Kontakt mit anderen Objekten realistisch simuliert werden kann. Einige Modelle stellen das Volumen der Hand nur grob durch primitive Formen nach [34]–[37] und sind damit hauptsächlich nur für akkuratere Rekonstruktion von Händen aus Tiefendaten oder Motion Capture gedacht. Andere Modelle verfolgen einen lernbasierten Ansatz zur Modellierung von Form und Pose der Hand durch die Deformation einer Vorlage [38]–[40]. Das MANO Modell [39] war eines der ersten Modelle dieser Art und wird in Sektion 3.4.1 noch einmal aufgegriffen. Komplexere Modelle wie das PIANO Modell [41] erweitern dieses Konzept durch die Einbindung von Knochendaten in den Trainingsprozess und das NIMBLE Modell [42] repräsentiert Hände über eine ineinandergreifende Modellierung von Hautoberfläche, Knochen und Muskeln und generiert zusätzlich noch realistische Texturen. Auch das NIMBLE Modell erfährt in Sektion 3.4.2 eine detailliertere Aufarbeitung.

### 2.4.1 Handposen Tools

Das Erzeugen realistischer Griffposen ist in der Animation häufig talentierten AnimatorInnen überlassen, es gibt aber auch prozedurale Ansätze zum Finden einer adäquaten Griffpose. GraspIt! [43] ist ein Tool aus der Robotik, das ein Interface zur Generierung von primitiven Griffposen bietet. Aufgrund der robotischen Natur sind die daraus entstehenden Griffposen allerdings entsprechend simpel und die Integration realistischer menschlicher Handmodelle ist schwierig.

Eine verhältnismäßig neue Anwendung, die erst während meiner Vorarbeit für diese Arbeit veröffentlicht wurde, nennt sich *Contact Edit* [44]. Das ist ein Tool zum intuitiven Modellieren von Hand-Objekt-Interaktionen. Dazu werden auf einer skelettierten Hand und auf dem Objekt Kontaktbereiche markiert und dann zugeordnet. Daraufhin wird mit Inverser Kinematik eine optimale Pose gefunden, die Kontaktflächen an der Hand auf die Kontaktfläche am Objekt anlegt. Allerdings ist *Contact Edit* als Tool für AnimatorInnen gedacht und lässt sich daher nicht einfach mit parametrischen Handmodellen zusammenführen.

## 2.5 Inverse Kinematik

Die Grundlagen Inverser Kinematik und damit auch den Großteil des Inhaltes in diesem Kapitel habe ich dem Buch *Computer Animation: Algorithms and Techniques* [45] entnommen. Einen konkreten Überblick über die Methoden und Formeln der Inversen Kinematik für Anwendungen in der Computergrafik habe ich mir zudem aus der Übersichtsstudie von Aristidou et al. [46] verschafft.

Kinematik kommt aus dem griechischen Wort für Bewegung. In der Computergrafik versteht man darunter häufig die Bewegung eines verbundenen Systems aus „Gelenken“ und „Knochen“, zum Beispiel die Bewegung der Extremitäten von menschlichen oder tierischen Figuren oder eines Roboterarms.

### 2.5.1 Prinzip, Funktionsweise, Anwendung

Ein System aus Gelenken und Gliedmaßen wie in Abbildung 2.9a nennt man einen Effektor. Ein Effektor hat eine hierarchische Struktur, beginnend beim Wurzelgelenk. Die absolute Position aller anderen durch Gliedmaßen angebotenen Gelenke ist abhängig von der Rotation so wie der Länge der Gliedmaße, die in der Hierarchie darüber liegen. Beschreibt man nun die Rotation jedes Gelenks relativ zu der seines

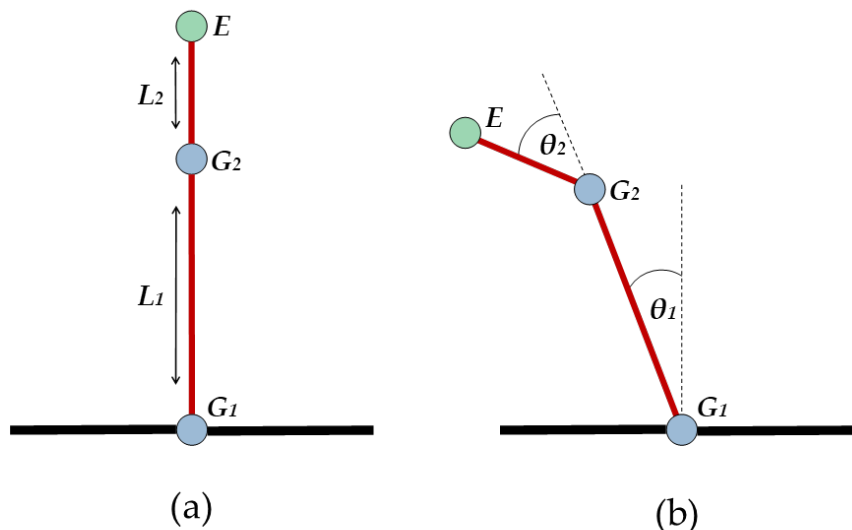


ABBILDUNG 2.9: (a) zeigt einen Effektor im Ausgangszustand mit den Längen der Gliedmaßen  $L_1$  und  $L_2$ . (b) zeigt eine Konfiguration des Effektors mit Gelenkwinkeln  $\theta_1$  und  $\theta_2$

Vorgängers entsteht eine Transformationskette mit der die absolute Position jedes Gelenkes recht einfach ausgerechnet werden kann. Nehmen wir an die Gelenke in Abbildung 2.9a können sich nur um die Achse senkrecht zur Papierebene drehen. Dann kann die Konfiguration des Effektors in Abbildung 2.9b einfach durch die zwei Winkel der zwei Gelenke beschrieben werden, angenommen die Länge der Gliedmaße sind bekannt.

Eine häufige Problemstellung ist jedoch nicht das Finden einer bestimmten Gesamtkonfiguration des Effektors sondern die Bewegung von einer Konfiguration hin zu einer Anderen. Spricht man zur Lösung des Problems von Vorwärtskinematik, so löst man es durch manuelle Spezifizierung der einzelnen Gelenkwinkel für Start-, Ziel und Zwischenkonfigurationen und interpoliert die einzelnen Gelenkwinkel von Konfiguration zu Konfiguration. Es ist allerdings bereits für kleine Effektoren schon sehr schwierig manuell eine Wunschkonfiguration zu erstellen, für sehr komplexe ist das schon praktisch unmöglich. Vor allem, weil die konkreten Werte der Gelenkwinkel häufig irrelevant sind und zum Beispiel nur die Position des Endgelenks ausschlaggebend ist. Wie zum Beispiel bei einem Roboterarm, der ein Objekt auf einem Tisch greifen soll. Je komplexer der Effektor ist, desto schwieriger ist es mit klassischer Vorwärtskinematik sinnvolle Konfigurationen zu finden. Die Methoden der Inversen Kinematik erlauben es eine gültige Konfiguration zu berechnen, gegeben einer Zielbedingung, wie zum Beispiel der Position eines oder mehrerer Endgelenke.

Es ist möglich einen Effektor als eine Funktion seiner Gelenkwinkel zu sehen. Stellt man diese Funktion nach der Zielbedingung um, so erhält man alle möglichen Konfigurationen, die die Bedingung erfüllen. Je komplexer der Effektor, desto komplizierter wird die Funktion und desto mehr mögliche Lösungen gibt es, aus denen man die "richtige" finden muss. Aufgrund der Komplexität von Effektoren in der Computergrafik sind daher numerische oder iterative Ansätze üblicher, da sie sich besser für große Effektoren skalieren lassen. Eine sehr verbreitete Methode ist

die schrittweise Approximation von einer Startkonfiguration aus zu einer Zielkonfiguration mithilfe der Jacobi-Matrix der Startkonfiguration. Diese Matrix besteht aus partiellen Ableitungen der einzelnen Gelenkwinkel und bildet die Änderung eines Gelenkwinkels auf eine Änderung der Endgelenkposition ab. Sie lässt sich als Ableitungsfunktion 2.5 der Gelenkwinkel  $\theta$  formulieren mit  $\mathbf{s}$  für die relevanten Parameter der  $k$  Endgelenke (als Beispiel hier die Position in  $x, y, z$  Koordinaten) und  $\theta$  für die  $n$  Gelenkwinkel.

Formel 2.6 zeigt, wie eine beliebige Änderung der Gelenkwinkel  $\Delta\theta$  mithilfe der Jacobi-Matrix auf eine Änderung der Endgelenkpositionen abgebildet werden kann. Da die Jacobi-Matrix lediglich partielle Ableitungen enthält, ist diese Abbildung nur für kleine  $\Delta\theta$  annähernd genau, da mit einer Änderung von  $\theta$  auch zwangsweise eine Änderung der Jacobi Matrix einhergeht.

$$J(\theta)_{ij} = \left( \frac{\partial s_i}{\partial \theta_j} \right) = \begin{pmatrix} \frac{\partial s_{x1}}{\partial \theta_1} & \frac{\partial s_{x1}}{\partial \theta_2} & \dots & \frac{\partial s_{x1}}{\partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s_{zk}}{\partial \theta_1} & \frac{\partial s_{zk}}{\partial \theta_2} & \dots & \frac{\partial s_{zk}}{\partial \theta_n} \end{pmatrix} \quad (2.5)$$

$$\Delta \mathbf{s} \approx J \Delta \theta \quad (2.6)$$

Angenommen  $t$  beschreibt die Zielpositionen der Endgelenke und  $s$  die aktuelle Position der Endgelenke. Die gewünschte Änderung der Endgelenkposition ist dann  $v = t - s$ . Da es das Ziel ist eine Änderung der Endgelenkpositionen  $\Delta \mathbf{s}$  zu finden, die möglichst nah an  $v$  herankommt, können wir  $v$  in Formel 2.6 einsetzen und nach der Änderung der Gelenkwinkel  $\Delta\theta$  umstellen(2.7)

$$\Delta \theta \approx J^{-1} v \quad (2.7)$$

Die Jacobi-Matrix ist jedoch selten quadratisch und invertierbar. Für dieses Problem gibt es diverse Lösungsansätze die in ihrer Gänze hier nicht relevant sind. Einzig die *Damped Least Squares*-Methode wird in Kapitel 3 weiter behandelt.

Die Jacobi-Matrix wird aus den Gelenkwinkeln zu einem festen Zeitpunkt berechnet. Ändern sich die Gelenkwinkel, ändert sich auch die Jacobi-Matrix. Für sehr kleine Änderungen der Gelenkwinkel kommt die berechnete Änderung der Wahrheit recht nahe, sie wird aber mit zunehmend großer Änderung der Gelenkwinkel immer weniger akkurat. Daher nähert man sich mit kleinen Schritten einer Zielkonfiguration an und berechnet nach jedem Schritt die Jacobi-Matrix neu, bis die Endgelenke ihrem Ziel ausreichend nah sind oder gleiches über eine maximale Anzahl Schritte nicht erreicht werden konnte.



## Kapitel 3

# Methoden

Zur Evaluierung der Frage, ob 1. unterschiedlich komplexe Handmodelle und 2. die Länge des sichtbaren Handgelenkes einen Einfluss auf die Erkennungsfähigkeit eines Netzes bei Hand-Objekt Interaktionen haben, erzeuge ich drei Datensätze, die sich bis auf die genutzten Hand Meshes nicht unterscheiden. Ein erster Datensatz nutzt von MANO generierte Hände ohne künstlichen Armfortsatz. Zur Untersuchung der ersten Fragestellung wird ein zweiter Datensatz erzeugt, der von NIMBLE generierte Hände nutzt. Zur Untersuchung der zweiten Fragestellung wird ein dritter Datensatz erzeugt, der wie der Erste das MANO Modell nutzt, dieses allerdings um einen künstlichen Armfortsatz erweitert.

Zur Erzeugung dieser Datensätze habe ich mir eine Pipeline erstellt, die mir die Erstellung von großen Mengen an Daten erleichtert. In diesem Kapitel möchte ich einen Überblick über diese Pipeline und die Entstehung der Datensätze geben. Zuerst geht es um die Wahl der zu greifenden Objekte und die Markierung von Kontaktpunkten auf diesen Objekten. Im Anschluss geht es um die Funktionsweise der beiden Handmodelle und wie ich Sie mithilfe eines Optimierers und meines eigenen Poseneditors eine objektspezifische Griffpose generieren lasse. Zuletzt beschreibe ich den Aufbau und die Erzeugung der drei Datensätze mit BlenderProc2.



### 3.1 Die Pipeline

Ich beginne mit einer groben Übersicht über die einzelnen Bestandteile der Pipeline. Sie besteht aus 4 separaten Schritten, die in 3 unterschiedlichen Anwendungen ausgeführt werden.

1. **Blender:** Manuelle Markierung von 3 Sets an Kontaktpunkten auf jedem Objekt mittels fingerspezifischer Vertexfarben. Anschließender Export der markierten Objektinstanzen und der Vertexpositionen der gefärbten Vertices.
2. **PoseEditor:** Generierung einer realistischen, objektspezifischen Griffpose für jede Objektinstanz mit dem MANO und dem NIMBLE Modell in einer eigenen Python Anwendung.
  - (a) Manuelle Parametrisierung und Positionierung des Handmodelles für eine grob ausgerichtete Initialpose für jede Objektinstanz.
  - (b) Die Parameter des Handmodelles mit inverser Kinematik optimieren, so dass die Fingerspitzen auf den markierten Kontaktpunkten auf dem Objekt liegen. Wenn das Resultat nicht akzeptabel ist, die Initialpose oder sogar die Markierung in Blender anpassen und erneut optimieren. Anschließender Export der Hand-Meshes und ihrer relativen Position.

3. **BlenderProc:** Systematisches Zusammenfügen der generierten Hand-Meshes mit den dazugehörigen Originalen der Objekte und Platzierung in einer 3D Szene. Anschließendes Rendern der Szenen und Abspeichern der Ground Truth.

---

	cracker_box		mustard_bottle
	master_chef_can		sugar_box
	potted_meat_can		large_marker
	tomato_soup_can		

---

TABELLE 3.1: Alle genutzten YCB Objekte mit ihren Namen

## 3.2 Objektdatensatz

Um die Vergleichbarkeit bei der Erforschung von Methoden zu gewährleisten ist es üblich seine Methodik mittels eines Benchmarks zu evaluieren. Ein üblicher Benchmark im Bereich der Objektposenbestimmung sind die BOP-Challenges [15], [19], [47], die die Leistung eingereicherter Methoden im Bereich der Objekterkennung anhand von mehreren vorgegebenen Datensätzen messen. Einer dieser Datensätze ist der YCB-Video Datensatz [17], den die Autoren des DexYCB Datensatzes [20] mit realen Hand-Objekt Interaktionen erweitert haben. Um die Effektivität und den Effekt von meinen synthetischen Trainingsdaten gut vergleichen zu können und eine Erweiterung des DexYCB Datensatzes möglich zu machen, habe ich mich bei der Auswahl der Objekte auf einen Teil der dort genutzten Objekte beschränkt.

Die Objekte stammen aus einem umfangreicheren Datensatz für reale Objekte [48] zur Forschung in der Manipulation durch Roboter und bieten daher ein breites

Spektrum an Form und Textur sowie detailgetreue 3D Scans und Modelle. Calli et al, Xiang et al und Chao et al [17], [18], [20] nutzen diese Objektsammlung um einen annotierten Realbild Datensatz zum Training neuronaler Netze für die Objektposenbestimmung zu erzeugen.

In Tabelle 3.1 sind die Objekte aufgelistet, die ich für die Erzeugung meiner vergleichenden Datensätze genutzt habe. Es war mir aufgrund der zeitintensiven manuellen Komponenten meiner Pipeline nicht möglich in dem gegebenen Zeitrahmen alle 21 Objekte aus den YCB-V und DexYCB Datensätzen in meinen Datensatz zu integrieren.

### 3.3 Markierung der Kontaktpunkte

Ich verfolge den Ansatz, dass bei einem Griff die Fingerspitzen aller 5 Finger auf dem gegriffenen YCB-Objekt liegen müssen. Daher definiere ich mögliche Griffe für ein Objekt über 5 Kontaktpunkte auf dessen Oberfläche. Ähnliche Ansätze wie zum Beispiel Hasson et al. [27] definieren noch eine zusätzliche Kontaktfläche am Handballen. Insgesamt deckt mein Ansatz deshalb nur einen kleinen Griffposen-Raum ab. Posen, bei denen nicht alle Finger auf dem Objekt liegen oder Finger das Objekt an mehr Stellen berühren, können mit diesem Ansatz nicht aktiv modelliert werden. Ich wähle diesen Ansatz aber, weil er mir die Posenkonstruktion merklich erleichtert. Ich vermute zudem, dass sich die Ergebnisse dieser Arbeit auch auf die an dieser Stelle ausgelassenen Griffposen übertragen lassen würden, da es hier vornehmlich um die Bestimmung einer Objekt- und nicht einer Handpose geht.

Die Kontaktpunkte der Fingerspitzen auf dem Objekt lege ich manuell in einer 3D Software fest. Ich habe mich dabei für Blender entschieden, weil es kostenlos ist, eine sehr nutzerfreundliche, detaillierte Bearbeitung von 3D Modellen ermöglicht und ich damit schon oft gearbeitet habe. Außerdem besitzt Blender eine integrierte Python Umgebung mit der die meisten Operationen automatisiert werden können. Im folgenden erläutere ich die einzelnen Schritte vom Laden der Objekte bis hin zum Export zur Weiterverarbeitung.

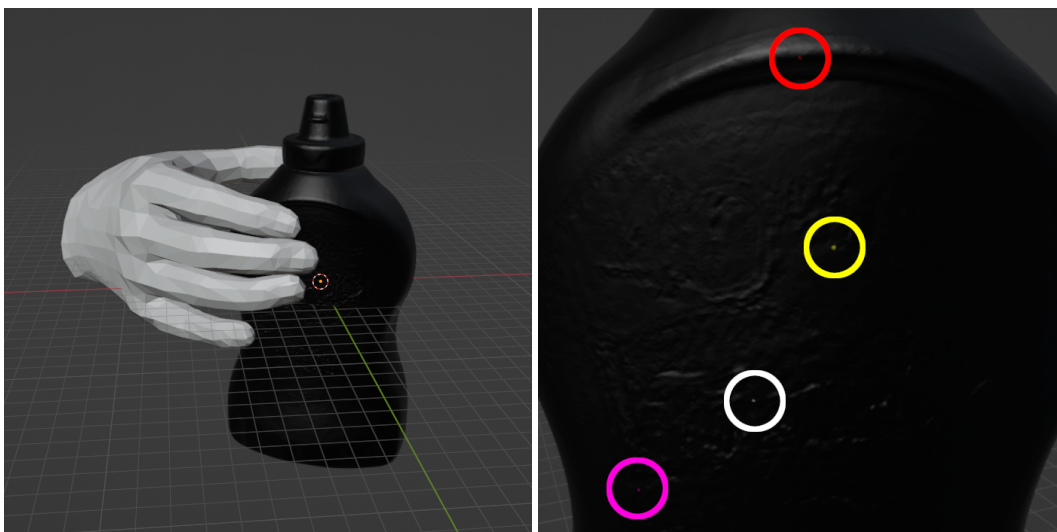


ABBILDUNG 3.1: Links eine plausible Initialpose. Rechts die markierten Kontaktpunkte auf dem Objekt für vier der Finger (mit Kreisen hervorgehoben). Der Kontaktpunkt für den Daumen ist auf der Rückseite des Objektes.

Zuerst lade ich eine Instanz des Objektes in Blender und färbe alle Vertices im *Vertex Paint Mode* schwarz. Um bei der Platzierung der Kontaktpunkte die Größenverhältnisse zwischen Objekt und Hand vor Augen zu haben, lade ich zusätzlich noch ein MANO Mesh in einer neutralen Pose. Dieses Mesh positioniere ich in etwa so, wie ich mir die spätere Griffpose vorstelle. Dann schätze ich nach Augenmaß ab, wo die Fingerspitzen der Hand beim Greifen auf dem Objekt liegen könnten und färbe an den geschätzten Stellen auf dem Objekt einen passenden Vertex mit einer fingerspezifischen Farbe. Abbildung 3.1 zeigt das Resultat für eines der Objekte. Im Anschluss lese ich mit einem Python Script in Blender die Positionen der eingefärbten Vertices aus und schreibe sie nach Farbe geordnet in eine Datei. Zusätzlich exportiere ich noch das eingefärbte Modell, um daran im nächsten Schritt die Hand für die Initialpose ausrichten zu können.

Für meine Zwecke habe ich diesen Prozess für jedes genutzte Objekt dreimal durchgeführt und somit für jedes Objekt drei unterschiedlich annotierte Objektinstanzen mit unterschiedlichen Kontaktpunkten erzeugt. Der gesamte Markierungsprozess für eine Objektinstanz dauert mit genug Übung etwa 15-20 Minuten.

### 3.4 Handmodelle

Zur Modellierung der unterschiedlichen Hände habe ich mich für die zwei parametrischen Handmodelle MANO [39] und NIMBLE [42] entschieden. Das NIMBLE Modell ist effektiv eine Weiterentwicklung des MANO Modells. Die beiden Modelle unterscheiden sich konkret vor allem in drei Aspekten:

1. **Textur:** NIMBLE generiert neben einem 3D Modell eine realistische Textur der Haut. Das MANO Modell erzeugt lediglich ein einfarbiges 3D Modell.
2. **Genauigkeit:** Die MANO Meshes bestehen aus nur knapp 800 Vertices, NIMBLE Meshes dagegen bestehen aus fast 5000 Vertices.
3. **Komplexität:** Die trainierte Deformation basiert bei MANO lediglich auf äußerlichen Scans verschiedener Handposen. NIMBLE nutzt hier zusätzlich noch MRT Scans um weitere anatomische Zusammenhänge modellieren zu können.

Beide Modelle erlauben durch ihre parametrische Natur sowohl die manuelle Bearbeitung der Posen- und Form-Parameter als auch die prozedurale Optimierung dieser Parameter. Im Falle von NIMBLE gilt das ebenfalls für die Parameter der Hauttextur. Außerdem vereinfacht das realistische Modell der Handoberfläche die Simulation von Hand-Objekt Interaktionen. Die Vollständigkeit dieser Modelle macht es für mich möglich durch das gezielte Spezifizieren, Optimieren und Randomisieren einiger weniger zentraler Parameter eine Vielzahl von verschiedenen, greifenden Händen zu generieren.

Um bei der Textur der Modelle Diversität sicherzustellen, aber trotzdem Vergleichbarkeit der Datensätze zu erhalten, habe ich mich entschieden die Textur-Parameter des NIMBLE Modells nicht zu randomisieren, sondern einen Satz aus fünf unterschiedlichen Parametervektoren zu nutzen. Äquivalent dazu habe ich fünf Farbtöne für die einfarbigen MANO Modelle definiert. Bei der Suche nach diversen Hautfarben habe ich mich an den fünf Hautstufen aus dem Unicode Standard zu Diversität [49] orientiert und versucht fünf repräsentative Parametervektoren für NIMBLE zu finden. Die fünf Farbtöne für MANO habe ich dann den fünf entstandenen NIMBLE Texturen nachempfunden.

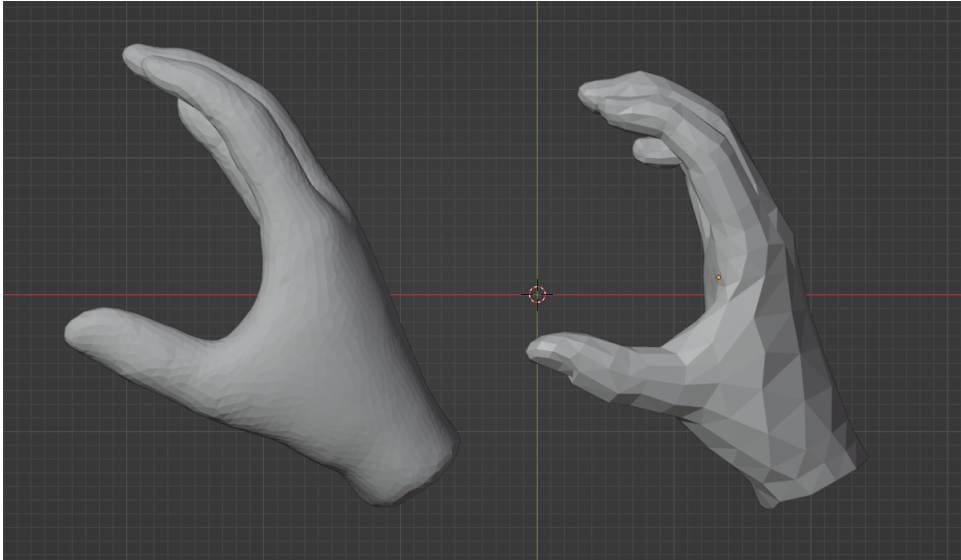


ABBILDUNG 3.2: Links ein 3D Modell vom NIMBLE Modell mit fast 5000 Vertices, rechts ein 3D Modell vom MANO Modell mit nur knapp 800 Vertices.

### 3.4.1 MANO

MANO ist eine Abkürzung für *hand Model with Articulated and Non-rigid defOrmations*. MANO [39] wurde als Erweiterung des SMPL Modells [50] für ganze Körper geschaffen, um die im Verhältnis zum Körper sehr kleinen aber komplexen Hände separat zu modellieren. MANO kann daher als alleinstehendes Modell für Hände genutzt werden.

Für den Trainingsdatensatz haben die Autoren 31 Testpersonen bis zu 51 unterschiedliche Handgriffe ausführen lassen und die Hände mit einem 3D Scanner aufgenommen. Das Modell wurde dann darauf trainiert eine generalisierte Handvorlage mittels Deformation an den insgesamt 2018 Scans im Datensatz auszurichten. Die Scans enthalten zwar Texturen der Hände, das MANO Modell beschränkt sich allerdings auf eine Approximation der Hand-Form und -Pose.

MANO deckt zwar einen gewissen Posen- und Formenraum ab, der Gesamtumfang an möglichen Posen und Formen ist aufgrund der Komplexität von Händen aber noch deutlich größer. Das MANO Modell modelliert die Pose aus Gründen der Einfachheit mit 15 Kugelgelenken (siehe Abb. 3.3). Offensichtlich können sich die Gelenke einer echten Hand nicht vollständig um drei Rotationsachsen drehen. Wie in Sektion 2.4 erläutert, haben die meisten Gelenke in der Hand nämlich weniger Freiheitsgrade, wodurch MANO überdimensioniert ist. Es lässt sich also nicht vermeiden, dass MANO unrealistische Ergebnisse erzeugt, wenn es Hände außerhalb seines gelernten Posen und Formen-Raums generieren muss.

Zum Erzeugen eines Meshes erwartet das MANO Modell im wesentlichen zwei Parametervektoren. Einen für die Form der Hand und einen für die Pose. Der Parameterraum für die 10 Formparameter ergibt sich aus der Hauptkomponentenzerlegung (PCA) aller personenspezifischen Vorlagen im Datensatz und vereint damit die Varianz der Form in den Trainingsdaten in 10 aussagekräftigen Parametern.

Für die Posen-Parameter ist es möglich entweder drei Winkelparameter pro Gelenk (15x3) anzugeben oder sie, wie bei der Form, in einem durch die PCA reduzierten Parameterraum (mit wählbarer Parameteranzahl) anzugeben. Ich habe mich dafür

entschieden in meiner Anwendung die überparametrisierte Variante mit konkreten Gelenkwinkel-Parametern zu verwenden, da ich damit die besten Ergebnisse produzieren konnte. Außerdem erleichtert die direkte Manipulation der Gelenkwinkel das manuelle Anpassen von Initialposen. Die PCA Repräsentation wird auch von Hampali et al [26] als nicht ausdrucksstark genug für ihren Zweck beschrieben, was mich in meiner Entscheidung bestätigt hat.

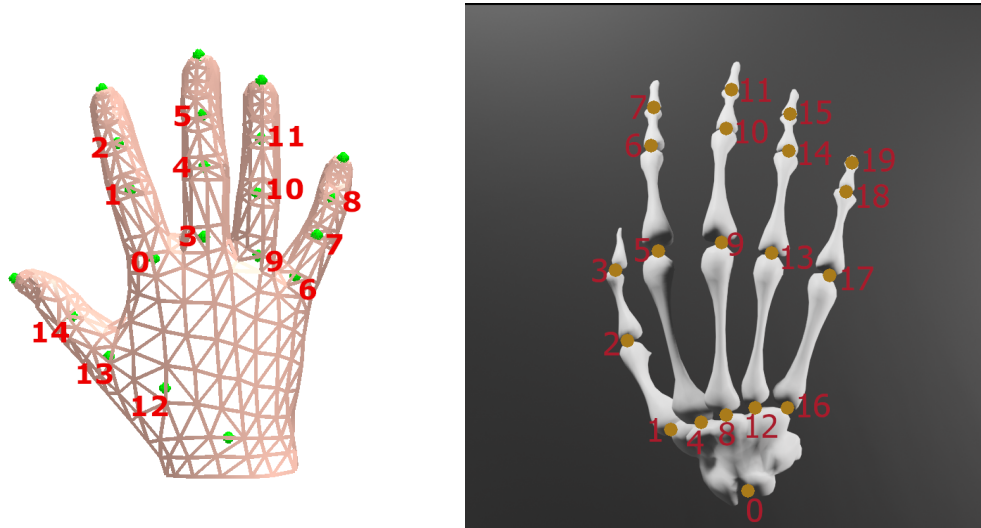


ABBILDUNG 3.3: Links die Gelenke im MANO Modell, hier als grüne Sphären; Rechts die 20 Gelenke des NIMBLE Modells, gezeigt an einem von NIMBLE generierten Knochen-Modell. Die Nummerierung entspricht jeweils der Reihenfolge im Parametervektor

### 3.4.2 NIMBLE

NIMBLE [42] steht für *Non-rIgid hand paraMetric modeling on Bone and MuscLE*. NIMBLE greift das Konzept von MANO auf und erweitert es mit der Modellierung von Muskeln und Knochen um Teile der realen Handanatomie. Einen Vorteil bietet dies laut den Autoren vor allem, weil die biomechanischen Wechselwirkungen zwischen Muskeln und Knochen einen nicht zu vernachlässigen Einfluss auf das äußerliche Erscheinungsbild der Hand haben. Um diese komplexen Zusammenhänge in eine Ground Truth zusammenzufassen, nutzen sie einen Datensatz aus 200 MRT Scans von 35 Händen von Testpersonen in bis zu 50 verschiedenen Posen. Für die Parametrisierung der Textur leiten die Autoren ein Modell aus einem separaten Datensatz aus 38 fotorealistischen Texturen ab. Im Gegensatz zum MANO Modell gibt das NIMBLE Modell auch Meshes für die modellierten Knochen und Muskeln aus. Für die Knochen und Muskel Meshes habe ich allerdings keine Verwendung, da ich nur an einer äußerlich realistischen Hand interessiert bin.

Der Aufbau der Parameter von NIMBLE ist ähnlich wie bei MANO. Die Form der Hand ist über 20 Parameter parametrisiert, gegeben durch eine PCA der Vorlagehände. Die Pose der Hand kann wie bei MANO direkt über 3 Gelenkwinkel pro Gelenk parametrisiert werden, oder über eine wählbare Anzahl an PCA Komponenten. Zu beachten ist hier, dass der Parameterraum fünf Gelenke mehr beschreibt, als der in MANO (siehe 0,4,8,12,16 in Abb. 3.3 rechts). Durch den anatomisch korrekten



Ansatz modelliert NIMBLE für jeden Mittelhandknochen der vier Finger ein zusätzliches Gelenk. Im Gegensatz zu MANO ist außerdem die globale Orientierung des Handgelenks im Parametervektor für die Pose enthalten. Zusätzlich gibt es noch 10 Parameter für die Textur, die ebenfalls die 10 Hauptkomponenten einer PCA des Texturmodells bilden.

Ich habe mich hier aus den selben Gründen wie bei MANO für die direkte Spezifizierung der Gelenkwinkel entscheiden. Während meiner Experimente habe ich allerdings festgestellt, dass sowohl die Rotation des Handgelenks, als auch die Bewegung der Mittelhandknochen (0,4,8,12,16) für meine Zwecke praktisch irrelevant sind. Die separate Bewegung der Mittelhandknochen ist nur für kleinste Werte tatsächlich realistisch, weshalb ich diese Parameter nicht aktiv nutze und optimiere. Die Orientierung des Handgelenks simuliere ich mit der globalen Orientierung des gesamten Meshes und konnte damit trotzdem gute Ergebnisse erzielen.

Auf meinem System mit einem intel i7 13700KF Prozessor ohne GPU Beschleunigung dauerte ein einzelner Durchlauf des NIMBLE Modells mit über zwei Sekunden signifikant länger als ein einzelner Durchlauf des MANO Modells. Durch einige Experimente stellte ich fest, dass das Deaktivieren eines Flags zur Kollisionsüberprüfung die Ausführungszeit eines Durchgangs auf fünf Hundertstel Sekunden reduziert und dabei kaum ein visueller Unterschied zu erkennen ist.

### 3.5 MinimalIK

Um von einer groben initialen Handpose zu einer greifenden Pose mit Objektkontakt zu gelangen nutze ich einen Optimierer für Inverse Kinematik. Den Algorithmus habe ich von Minimal-IK [51] übernommen, einem simplen Optimierer für das MANO Modell, habe ihn allerdings von der integrierten MANO Implementation entkoppelt und eigene Schnittstellen für MANO und NIMBLE implementiert.

Das Ziel ist es, eine Pose zu finden, für die die Fingerspitzen auf den annotierten Kontaktpunkten des Objektes liegen. Ähnlich der Markierung von Kontaktpunkten auf den Objekten definiere ich daher für die Handmodelle pro Fingerspitze einen Vertex, der in der finalen Pose auf dem entsprechenden Objektvertex liegen soll. Abb. 3.4 zeigt welche Vertices ich bei MANO und NIMBLE für diesen Zweck ausgewählt habe.

Die von mir implementierten Schnittstellen für beide Modelle geben bei Ausführung des Modells mit übergebenen Parametern die resultierende Position dieser fünf Vertices in Objektkoordinaten zurück.

Der Minimal-IK Optimierer nutzt die *Damped least squares* Methode, auch *Levenberg-Marquardt Algorithmus* genannt, zur Lösung der Inversen Kinematik. Gleichung 3.1 zeigt wie dieses Verfahren das Problem der schwer invertierbaren Jacobi-Matrix umgeht. Wichtig ist hier vor allem der Dämpfungsparameter  $u$  mit dem Stabilität gegen Konvergenzgeschwindigkeit getauscht werden kann. Die Implementierung von Minimal-IK passt den Dämpfungsparameter zusätzlich in jedem Iterationsschritt an um diese beiden Faktoren effektiver gegeneinander auszuspielen.

$$\Delta\theta = (J^T J + uI)^{-1} J^T v \quad (3.1)$$

Die Jacobi-Matrix wird aus simplen Approximationen der partiellen Ableitungen zusammengesetzt. Für die Ableitung eines Posenparameters  $p_k$  aus  $\theta$  wird das

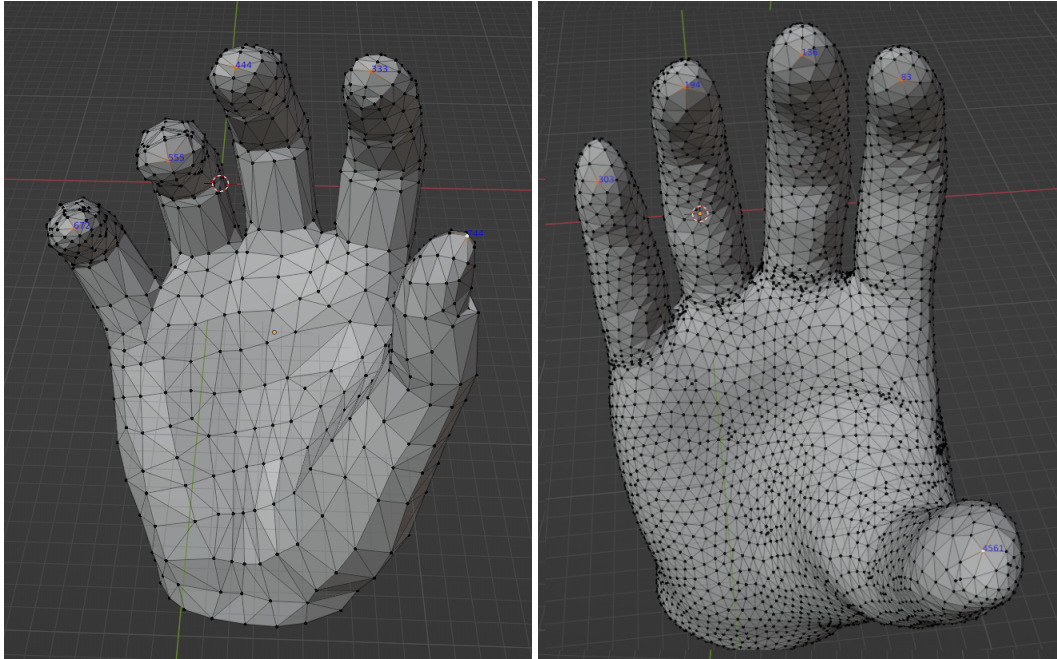


ABBILDUNG 3.4: Links ein MANO Modell und rechts ein NIMBLE Modell, jeweils mit den für die Optimierung ausgewählten Vertices und deren Indizes

gesamte Modell  $M$  über die Schnittstelle zwei mal mit  $p_k \pm \epsilon$  ausgeführt. Eine einzelne Spalte der Jacobi-Matrix  $J_k$  setzt sich dann aus dem Differenzquotienten der resultierenden Fingerspitzenpositionen zusammen (Gleichung 3.2).  $\epsilon$  ist ein steuerbarer Parameter, den ich allerdings nie anpassen musste. Der Standardwert in Minimal-IK ist  $1e^{-5}$ .

$$J_k = \frac{M(\theta_{+\epsilon}) - M(\theta_{-\epsilon})}{2\epsilon} \quad (3.2)$$

Im Posen Editor transformiere ich die Koordinaten der Kontaktpunkte zuerst in die Objektkoordinaten der Handmodelle und nutze dann die hier vorgestellte Optimierungsmethode um die Handpose in Handkoordinaten zu optimieren.

Der Optimierer ist verhältnismäßig einfach und stark abhängig von einer gut gewählten Initialpose. Meine Experimente haben ergeben, dass der Optimierer die Beugung der vier Finger dabei etwas besser modellieren kann als die Bewegung des Daumens.

### 3.5.1 Besonderheiten für NIMBLE

Für die oben beschriebene Methode müssen die Modelle für jeden einzelnen Parameter zweimal ausgeführt werden. Für MANO sind das pro Iteration  $15 \times 3 \times 2 = 90$  Modellaufrufe. Mit einer Laufzeit von ein paar Hundertstel Sekunden pro Aufruf und maximal 20 Iterationen kommt man für MANO auf eine akzeptable maximale Laufzeit von einigen wenigen Sekunden.

Um die Gesamtlaufzeit für NIMBLE im niedrigen Minutenbereich zu halten, muss ich das in 3.4.2 erwähnte Flag zur Kollisionsbehandlung weglassen. Da eine einzige Iteration  $20 \times 3 \times 2 = 120$  Modellaufrufe erfordert, würden 2.5 Sekunden Laufzeit pro Modellaufruf fast fünf Minuten pro Iteration bedeuten. Es ist möglich, dass durch die fehlende Kollisionsbehandlung während der Optimierung unnötig



unrealistische Posen entstehen. Ich konnte allerdings während meiner Experimente keinen konkreten Unterschied zwischen den Resultaten mit und ohne Kollisionsbehandlung feststellen. Die groben, aber doch recht akkuraten Initialposen verhindern hier vielleicht, dass es während der Optimierung überhaupt zu großen Kollisionen kommt.

MANO und NIMBLE sind außerdem unterschiedlich dimensioniert. Da die Methode von Minimal-IK mit einem absoluten Fehlerterm in Objektkoordinaten arbeitet, muss der Fehler Schwellwert für das größer dimensionierte NIMBLE Modell entsprechend angepasst werden.

## 3.6 Posen Editor

Um die markierten Objekte mit den Handmodellen zusammenzubringen und mithilfe des Optimierers eine Hand-Objekt Interaktion zu simulieren, habe ich einen visuellen *Posen Editor* implementiert. Er erlaubt mir die Parameter der beiden Modelle manuell anzupassen und die Hand Meshes relativ zu den Objekt-Modellen auszurichten, um eine gute Initialpose für den Optimierer zu finden. Der Optimierungsprozess lässt sich ebenfalls über diese Anwendung starten und überwachen. [Abbildung 3.5](#) zeigt einen Screenshot dieses Editors.

Als Grafik-Engine nutze ich dafür die Python-Engine *Panda3D* [52]. Ich habe mich für eine native Python-Engine entschieden, weil beide Handmodelle in Python implementiert sind und die Einbindung der für die Modelle erforderlichen Python-Bibliotheken in eine externe Engine unplanbar viel Extraaufwand bedeutet hätte. Blender hätte aufgrund seiner eingebauten Python-Umgebung, der großen Menge an Features und der Tatsache, dass die Vorverarbeitung der Objekte ebenfalls in Blender stattfindet, auch viele Vorteile geboten. Allerdings hat sich hier die Einbindung der benötigten Python Bibliotheken ebenfalls als schwierig herausgestellt. MANO wurde im Original im veralteten Python 2 geschrieben. Deshalb verwende ich eine alternative Implementierung von MANO für Python 3 [53].

### 3.6.1 Funktionen

[Abbildung 3.5](#) erklärt die Benutzeroberfläche des Posen Editors. In der 3D Szene befinden sich jeweils eine MANO und eine NIMBLE Hand so wie ein annotiertes Objekt. Außerdem noch eine Grundebene zur einfacheren Orientierung im Raum. Das NIMBLE Modell ist um einen Faktor von 0.22 skaliert, um es an die Größe des MANO Modells und der Objekte anzupassen. Diesen Wert habe ich durch Experimente festgelegt. Das annotierte Objekt bleibt zur Laufzeit gleich und ist fest am Ursprung der Szene verankert. Das bedeutet für jede annotierte Objektinstanz muss die Anwendung neu gestartet werden.

Um eine passende Initialpose zu definieren beginne ich mit dem groben Ausrichten der MANO Hand über die Textfelder [3.5\(6\)](#). Häufig kann ich hierfür auch bereits früher gespeicherte Konfigurationen laden [3.5\(7\)](#) und als Startpunkt für eine neue Konfiguration nutzen. Die initiale Griffpose der Hand ist häufig die selbe, da der Optimierer hier einen Großteil der Details übernimmt. In [Abbildung 3.2](#) ist diese initiale Griffpose für beide Handmodelle zu sehen. In Einzelfällen war hier allerdings zusätzlich eine manuelle Anpassung nötig um Anomalien des Optimierers vorzubeugen.

Zuletzt wird der Optimierer gestartet [3.5\(5\)](#). Das Ergebnis wird direkt im Editor angezeigt. Wenn die Fingerspitzen auf den Kontaktpunkten liegen und der Griff

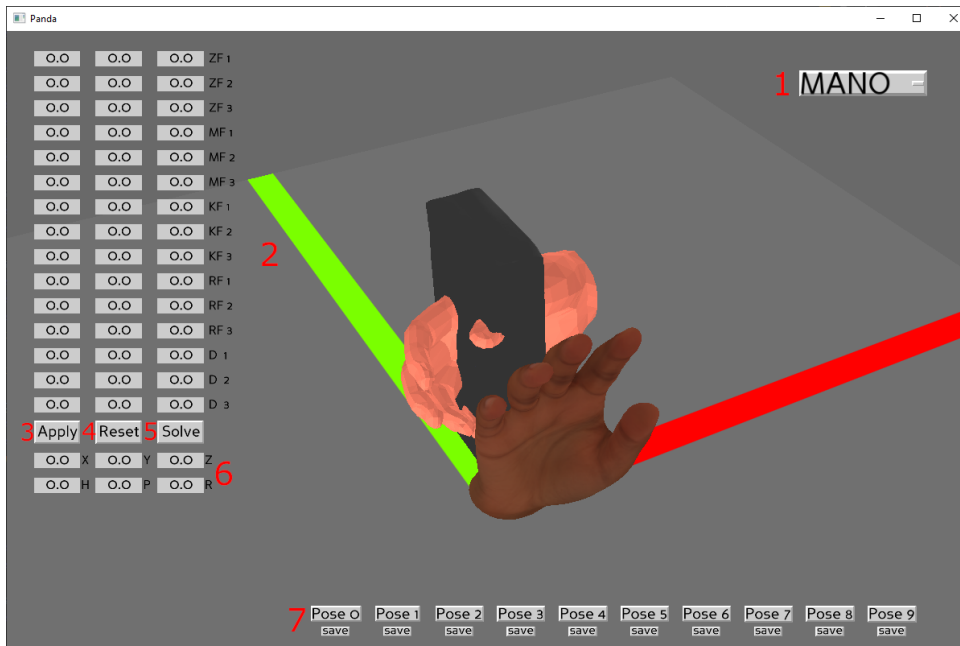


ABBILDUNG 3.5: 1 Wechseln des aktiven Handmodells; 2 Anpassbare Parameter des aktiven Handmodells; 3 Ausführen des aktiven Handmodells mit obigen Parametern; 4 Alle Parameter auf 0 setzen; 5 Starten der Optimierung mit Inverser Kinematik; 6 Globale Position und Rotation des aktiven Handmodells; 7 Zehn Slots zum Speichern und Laden von Posenkonfigurationen für aktives Handmodell;

insgesamt realistisch aussieht, wiederhole ich den Prozess für das NIMBLE Modell. Dabei lege ich besonderen Wert darauf, dass die Initialpose so nah wie möglich an die Initialpose für das MANO Modell herankommt, um ein möglichst ähnliches Posenprofil durch den Optimierer zu erhalten. Für beide Handmodelle gilt, dass ab und zu mehrere Iterationen von Ausrichtung und Optimierung nötig sind um wünschenswerte Ergebnisse zu produzieren.

Im Hintergrund speichert das Programm nach der abgeschlossenen Optimierung die finale Instanz des Hand Meshes, dessen Position und Rotation, so wie die Modell-Parameter der Initialpose und der optimierten Pose. Um etwas mehr Posen Varianz in den Datensatz zu bringen wird die Optimierung automatisch weitere vier Male wiederholt, allerdings mit leicht veränderter Relativposition und Rotation und einem randomisierten Parametervektor für die Form (Im Falle von NIMBLE zusätzlich mit jeweils einem der fünf vordefinierten Parametervektoren für die Textur). Das Resultat ist eine Sammlung aus fünf leicht unterschiedlichen MANO/NIMBLE Händen für jedes Set an Kontaktpunkten eines Objektes mit. Dieser Prozess muss für jede Objektinstanz wiederholt werden und dauert pro Instanz und mit genug Übung durchschnittlich 10-20 Minuten.

### 3.7 BlenderProc2

Um die drei Bilddatensätze zu erzeugen müssen die generierten Handposen mit den texturierten YCB-Objekten zusammengeführt und in einer Szene platziert und gerendert werden. Ich nutze dafür BlenderProc2 [54], eine anpassbare Pipeline zur Erzeugung von fotorealistischen RGB Datensätzen über eine einfache Blender-Python

Schnittstelle.

Da ich meine synthetischen Datensätze auf realen Daten von DexYCB [20] evaluieren möchte, erzeuge ich mit meiner BlenderProc2 Pipeline Szenen, die grob dem Versuchsaufbau von DexYCB ähneln. Dabei orientiere ich mich aber am Schema der *Domain Randomization* für synthetische Daten und randomisiere Hintergründe, Objektposen und Lichtverhältnisse. Abb. 3.6 zeigt ein Beispiel aus dem DexYCB Datensatz und im Vergleich dazu ein von mir erzeugtes synthetisches Bild.



ABBILDUNG 3.6: Links ein Bild aus dem DexYCB Datensatz [20], rechts ein mit meiner Pipeline generiertes Bild.

Die Szenerie für meine synthetischen Bilder orientiert sich an einem von BlenderProc2 bereitgestellten Beispiel zur Erzeugung eines synthetischen Datensatzes für die BOP-Challenge [15].

Zu Beginn wird zufällig eines der sieben von mir verwendeten YCB-Objekte ausgewählt, so wie eines der drei dazugehörigen Kontaktpunktesets. Zu diesem Kontaktpunkteset wurden fünf leicht unterschiedliche Hand Meshes generiert, von denen ebenfalls zufällig eines ausgesucht wird. Außerdem werden aus den 21 verfügbaren YCBV-Objekten zufällig einige weitere *Ablenkungsobjekte* ausgesucht.

Die Szene befindet sich innerhalb eines würfelförmigen Raumes. Die 4 Innenwände so wie der Boden erhalten für jede Szene ein zufälliges PBR Material aus einer Sammlung von tausenden frei zugänglichen Materialien. Die Decke des Würfels ist eine licht-emittierende Ebene, deren Leuchtstärke und Farbe ebenfalls für jede Szene randomisiert werden. Die aus dem Posen-Editor exportierte Relativpose der Hand-Meshes wird genutzt um die Hand am zu greifenden Objekt auszurichten. Dieses Hand-Objekt Konstrukt wird dann an einer zufälligen, schwebenden Position mit zufälliger Rotation im Raum platziert. Zusätzlich werden die Ablenkungsobjekte mit einer Physik-Simulation aus ähnlicher Höhe auf den Boden fallen gelassen. Jede Szene wird aus mehreren randomisierten Kamerawinkeln gerendert, immer mit dem gegriffenen Objekt im Fokus.

Da alle drei Datensätze die selben Szenen zeigen sollen (siehe Abb. 3.7), nur mit einem alternativen Handmodell, werden alle zufällig generierten Szenenparameter nach einem ersten Durchlauf abgespeichert um für folgende Durchläufe mit einem anderen Handmodell wieder geladen werden zu können. Diese Logik konnte ich aus einem Script von meinem Betreuer Janis Roskamp übernehmen und für meine Zwecke erweitern.

### 3.7.1 MANO Vorverarbeitung und Armfortsatz

Das MANO Mesh aus dem Posen-Editor muss vor dem Rendern der Szene noch angepasst werden. Zum Einen wird dem farblosen Material der Hand eine der fünf vordefinierten Farben zugewiesen, entsprechend der zufällig gewählten Hand-Alternative. Zum Anderen sind die generierten MANO Meshes am Handgelenk offen. Dieses Loch wird mit einem einzelnen Polygon verschlossen. Für den dritten Datensatzes wird dieses Polygon einfach nur extrudiert, um einen Armfortsatz zu simulieren (siehe Abb. 3.7).

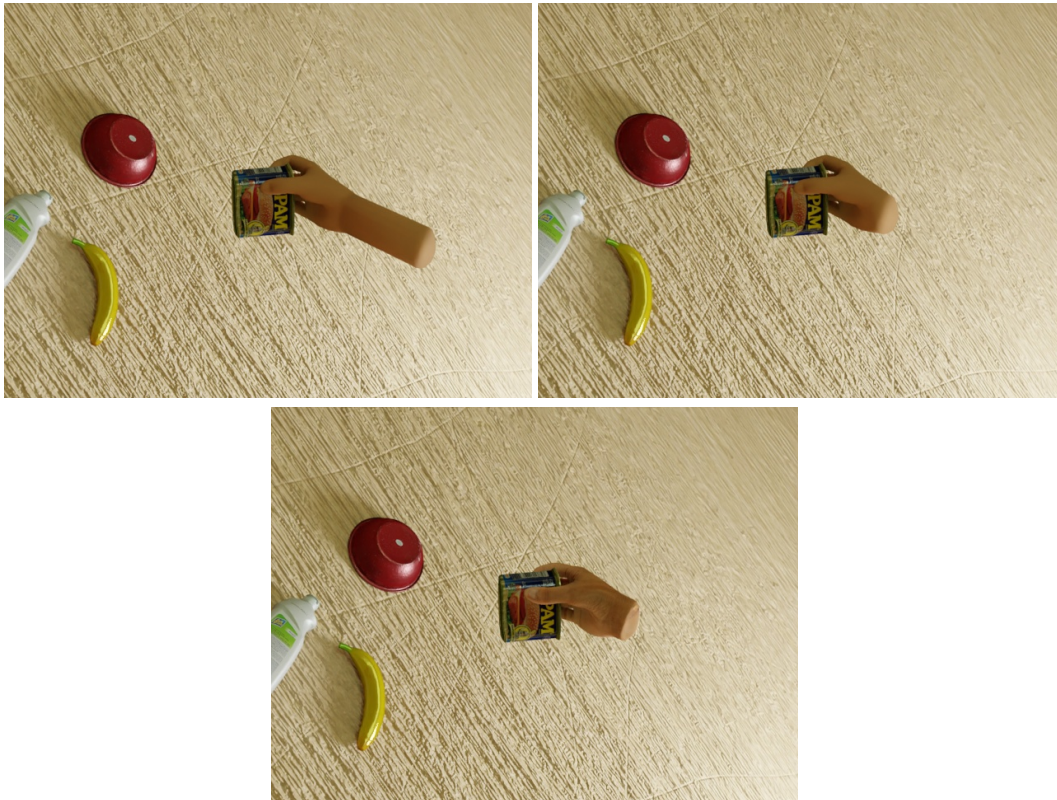


ABBILDUNG 3.7: Zu sehen ist die selbe Szene mit drei unterschiedlichen Händen. Dem MANO Modell mit Armfortsatz (oben links), dem MANO Modell ohne Armfortsatz (oben rechts) und dem NIMBLE Modell (unten)

## 3.8 Resultate

Ich erzeuge in mehreren separaten Schritten drei zentrale synthetische Datensätze von Hand-Objekt Interaktionen, die sich im Aufbau der Szenen und den Kontaktpunkten auf den Objekten gleichen. Unterschiedlich ist nur die Form und Treue der greifenden Hand.

Alle drei Datensätze bestehen aus 20k RGB-Bildern und typischen automatisch erzeugten Ground Truths, analog zum BOP-Benchmark, wie ein Tiefenbild, eine Maske und Szenenparameter wie Kamera und Objektposes. Die 20k Bilder zeigen 800 Szenen in denen 7 verschiedene YCB-Objekte auf drei mögliche Arten gegriffen werden. Für jeden möglichen Griff gibt es 5 Hand-Meshes, die sich vor allem in der Textur bzw. Farbe unterscheiden. Zusätzlich sind vier der fünf Hand-Meshes



mit randomisierten Form-Parametern generiert und zusätzlich mit einem leicht randomisierten Offset der Initialpose für die Inverse Kinematik versehen. Jede Szene setzt sich außerdem aus einem Raum mit zufälligem PBR-Material, einer randomisierten Beleuchtung und drei zufällig ausgewählten Ablenkerobjekte aus dem gesamten YCBV-Objektset. Alle Szenen werden aus 25 zufälligen Kameraperspektiven gerendert, bei denen allerdings immer der Fokus auf dem zu erkennenden Objekt bleibt. Das Rendern der Datensätze hat insgesamt knapp 30 Stunden gedauert.

## Kapitel 4

# Evaluation

Ich habe drei zentrale Datensätze erzeugt, die sich bis auf das genutzte Handmodell im Aufbau der Szene gleichen. Als minimale Baseline betrachte ich den Datensatz mit einer reinen MANO Hand, und vergleiche die Ergebnisse eines darauf trainierten Netzes mit denen für die beiden anderen Datensätze. Die MANO Hand ist das simpelste Modell, daher bietet es sich dafür am ehesten an.

Zur Evaluation der Datensätze nutze ich ein ZebraPose Netz [55]. ZebraPose belegte in der BOP-Challenge 2022 [15] Platzierungen in den Top 20 aller eingereichten Methoden und gehört damit zum State of the Art in der Bestimmung von 6D Objekt-*posen* aus RGB Bildern. Zudem ist ZebraPose öffentlich verfügbar auf GitHub. Ich lasse auf allen drei Datensätzen ( $M$ ,  $N$ ,  $A$ ) ein separates Netz trainieren. Als Ablationsstudie lasse ich auf einem vierten Datensatz ohne Hand-Objekt Interaktion ( $O$ ) trainieren. Dieser vierte Datensatz nutzt nicht die selben Szenen, arbeitet aber mit dem selben Set an Objekten. ZebraPose muss für jede Objektinstanz einzeln trainiert werden, in diesem Fall also für jedes Objekt aus jedem Datensatz. Bei 20k Bildern im Datensatz ergeben sich für jedes Objekt knapp 3k Bilder für das Training des Netzes. Von 80k Iterationen mit einer Batchsize von 32 wurde mittels eines Validierungssets die beste Iteration ausgewählt und für die Evaluation genutzt. Der Trainingsprozess für einen einzigen Datensatz mit allen 7 Objekten dauerte insgesamt knapp 74 Stunden auf einer RTX 3090.

Als Testdaten für die trainierten ZebraPose Netze wurden einige Videosequenzen für passende Objekte aus dem DexYCB Datensatz [20] verwendet. Damit evaluiere ich die vollständig synthetisch trainierten Netze auf realen Testdaten. Die Einzelbilder kommen aus jeweils acht unterschiedlichen Perspektiven über drei verschiedenen Videosequenzen und Versuchsaufbauten pro Objekt. Jede Videosequenz besteht aus rund 70 Frames, die sich in unmittelbarer Nähe zueinander stark ähneln. Daher habe ich analog zum Vorgehen der Autoren von DexYCB zur Evaluation nur jeden vierten Frame einer ganzen Videosequenz genutzt. Dadurch ergeben sich je nach Anzahl der Frames ca. 420 reale Testbilder pro Objekt. Abbildung 4.1 zeigt beispielhaft jeden siebten Frame aus einer der Videosequenzen.

### 4.1 Metriken

**ADD** beschreibt die durchschnittliche Distanz der Objektpunkte zwischen der Vorhersage und der Ground Truth. Üblicherweise gilt eine Pose als korrekt, wenn der Fehler hier unter einem Grenzwert von zehn Prozent des Objektdurchmessers liegt. Der Wert zeigt den Anteil der vorhergesagten Posen, deren Distanz zur Ground Truth unter diesen Schwellwert fällt. **ADD-S** funktioniert ähnlich wie die ADD, vergleicht allerdings die Distanz der Objektpunkte der Vorhersage mit dem nächstliegenden Nachbarn der Ground Truth. Das ist insbesondere relevant für Objekte mit Symmetrien, die mit mehreren verschiedenen Posen das gleiche Bild erzeugen. Alle



ABBILDUNG 4.1: Einzelne Frames einer Videosequenz aus DexYCB [20]

Objekte in meinen Datensätzen besitzen keine visuellen Symmetrien, da sie unterschiedliche Texturen haben. Trotzdem ist die ADD-S Metrik zum Beispiel für Greifaufgaben interessant, bei denen vor allem eine grobe Abschätzung der Form der Objekte im Raum wichtig ist. Die ADD-S wird daher immer größer ausfallen als die ADD. Eine hohe ADD deutet neben einer akkuraten Schätzung der räumlichen Ausdehnung auch auf eine visuell akkurate Schätzung hin. **AUC** beschreibt die *area under the curve* von der Kurve, die durch Grenzwerte auf der x-Achse und der resultierenden Präzision einer Metrik (ADD oder ADD-S) auf der y-Achse entsteht. Der Grenzwert hat hier analog zum Vorgehen im PoseCNN Paper [17] einen Wertebereich von null bis zehn Zentimetern. Mit diesem Wert und der namensgebenden Kurve lässt sich das Verhalten über unterschiedliche Grenzwerte beobachten. Dadurch ergibt sich ein Überblick über die Konsistenz der Ergebnisse.

## 4.2 Ergebnisse

### 4.2.1 Vergleich der Metriken

Objekt	ADD			ADD-S		
	M	N	O	M	N	O
cracker_box	<b>93.7</b>	90.5	83.3	<b>97.8</b>	96.4	96.1
large_marker	7.9	10.1	<b>12.0</b>	<b>32.7</b>	28.4	28.8
master_chef_can	47.9	<b>51.1</b>	38.8	88.8	<b>89.2</b>	86.2
mustard_bottle	<b>37.1</b>	34.7	36.4	68.5	<b>73.2</b>	66.9
potted_meat_can	52.1	48.8	<b>52.8</b>	81.6	80.5	<b>82.1</b>
sugar_box	36.2	<b>38.3</b>	33.6	<b>75.7</b>	74.7	71.9
tomato_soup_can	28.9	30.5	<b>39.5</b>	64.2	64.2	<b>68.1</b>
Mittel	<b>43.4</b>	<b>43.4</b>	42.3	<b>72.8</b>	72.4	71.5

TABELLE 4.1: Vergleich von ADD(-S): MANO(M) vs NIMBLE(N) vs ohne Handinteraktion(O). Pro Objekt ist jeweils der beste Wert einer Metrik über die drei Datensätze **fett** markiert.

Tabelle 4.1(links) zeigt wie die ADD im Vergleich von MANO und NIMBLE praktisch keinen Unterschied aufweist. Auch im Vergleich von MANO mit und ohne Armfortsatz (Tabelle 4.2(links)) zeigt sich hier kein merklicher Unterschied. In



Objekt	ADD			ADD-S		
	M	A	O	M	A	O
cracker_box	<b>93.7</b>	91.0	83.3	97.8	<b>98.8</b>	96.1
large_marker	7.9	7.7	<b>12.0</b>	<b>32.7</b>	31.7	28.8
master_chef_can	47.9	<b>45.0</b>	38.8	<b>88.8</b>	<b>88.8</b>	86.2
mustard_bottle	<b>37.1</b>	36.1	36.4	68.5	<b>69.5</b>	66.9
potted_meat_can	52.1	45.1	<b>52.8</b>	81.6	<b>82.1</b>	<b>82.1</b>
sugar_box	36.2	<b>41.1</b>	33.6	75.7	<b>81.8</b>	71.9
tomato_soup_can	28.9	35.8	<b>39.5</b>	64.2	<b>73.2</b>	68.1
Mittel	<b>43.4</b>	43.1	42.3	72.8	<b>75.1</b>	71.5

TABELLE 4.2: Vergleich von ADD(-S): MANO(M) vs MANO mit Armfortsatz(A) vs ohne Handinteraktion(O). Pro Objekt ist jeweils der beste Wert einer Metrik über die drei Datensätze **fett** markiert.

beiden Fällen ist der Datensatz ohne Handinteraktion nur minimal abgeschlagen. Betrachtet man die AUC der ADD Metrik in Tabelle 4.3(links) so zeigen sich jedoch auffällig gute Ergebnisse des Armfortsatz Datensatzes. Auch der NIMBLE Datensatz setzt sich hier sehr leicht vom MANO Datensatz ab, wenn auch nicht so stark wie der Datensatz mit Armfortsatz. Die verzeichnete Unterschiede befinden sich bei den einzelnen Objekten fast immer im niedrigen einstelligen Prozentbereich und fallen über alle Objekte gemittelt im Vergleich von MANO und NIMBLE nur kaum relevante 0.9% besser zugunsten von NIMBLE aus. Der Armfortsatz Datensatz setzt sich im Mittel mit zwei Prozent höherer Präzision allerdings schon deutlich ab. Der Datensatz ohne Handinteraktion liegt auch hier um ein Prozent abgeschlagen hinter allen anderen Datensätzen ohne ein einziges bestes Einzelergebnis.

Objekt	AUC_ADD				AUC_ADD-S			
	M	N	A	O	M	N	A	O
cracker_box	<b>87.6</b>	86.6	86.9	83.0	93.0	92.8	<b>93.2</b>	91.2
large_marker	41.5	41.4	<b>44.9</b>	38.7	54.8	53.7	<b>57.8</b>	48.6
master_chef_can	59.0	<b>62.6</b>	59.3	56.5	88.4	<b>90.0</b>	89.2	87.6
mustard_bottle	60.9	63.4	<b>63.6</b>	60.4	78.9	<b>82.6</b>	80.2	76.7
potted_meat_can	77.0	75.8	<b>77.1</b>	79.5	88.9	88.3	87.9	<b>89.1</b>
sugar_box	53.4	54.7	<b>56.8</b>	52.8	83.4	84.3	<b>85.7</b>	80.8
tomato_soup_can	65.5	67.0	<b>70.2</b>	67.6	79.4	81.5	<b>83.3</b>	79.9
Mittel	63.6	64.5	<b>65.6</b>	62.6	81.0	81.9	<b>82.5</b>	79.1

TABELLE 4.3: Vergleich der AUC\_ADD(-S) für alle vier Datensätze. Pro Objekt ist jeweils der beste Wert einer Metrik über alle Datensätze **fett** markiert.

Vergleicht man die symmetrische ADD Metrik zwischen den MANO und NIMBLE Datensätzen in Tabelle 4.1(rechts) so zeigt sich hier ebenfalls kein Unterschied. Allerdings setzt sich der Datensatz mit Armfortsatz eindeutig bei fast allen Objekten vom MANO Datensatz ab und erreicht ein im Mittel 2.3% besseres Ergebnis als der reine MANO Datensatz (Tabelle 4.2(rechts)). Der Datensatz ohne Handinteraktion liegt hier etwas deutlicher abgeschlagen hinter den drei zentralen Datensätzen als bei der strengeren ADD Metrik. Die AUC der ADD-S in Tabelle 4.3(rechts) zeichnet

ein praktisch gleiches Bild wie die AUC der ADD mit einem leichten Vorsprung für den Datensatz mit Armfortsatz von im Mittel 1.5% und gleichem Vorsprung von NIMBLE gegenüber MANO mit im Mittel 0.9%. Wie zu erwarten ist auch hier der Datensatz ohne Handinteraktion ohne ein einziges bestes Einzelergebnis leicht abgeschlagen, wenn auch nur um 2% im Mittel.

### 4.2.2 Präzision/Grenzwert Kurven

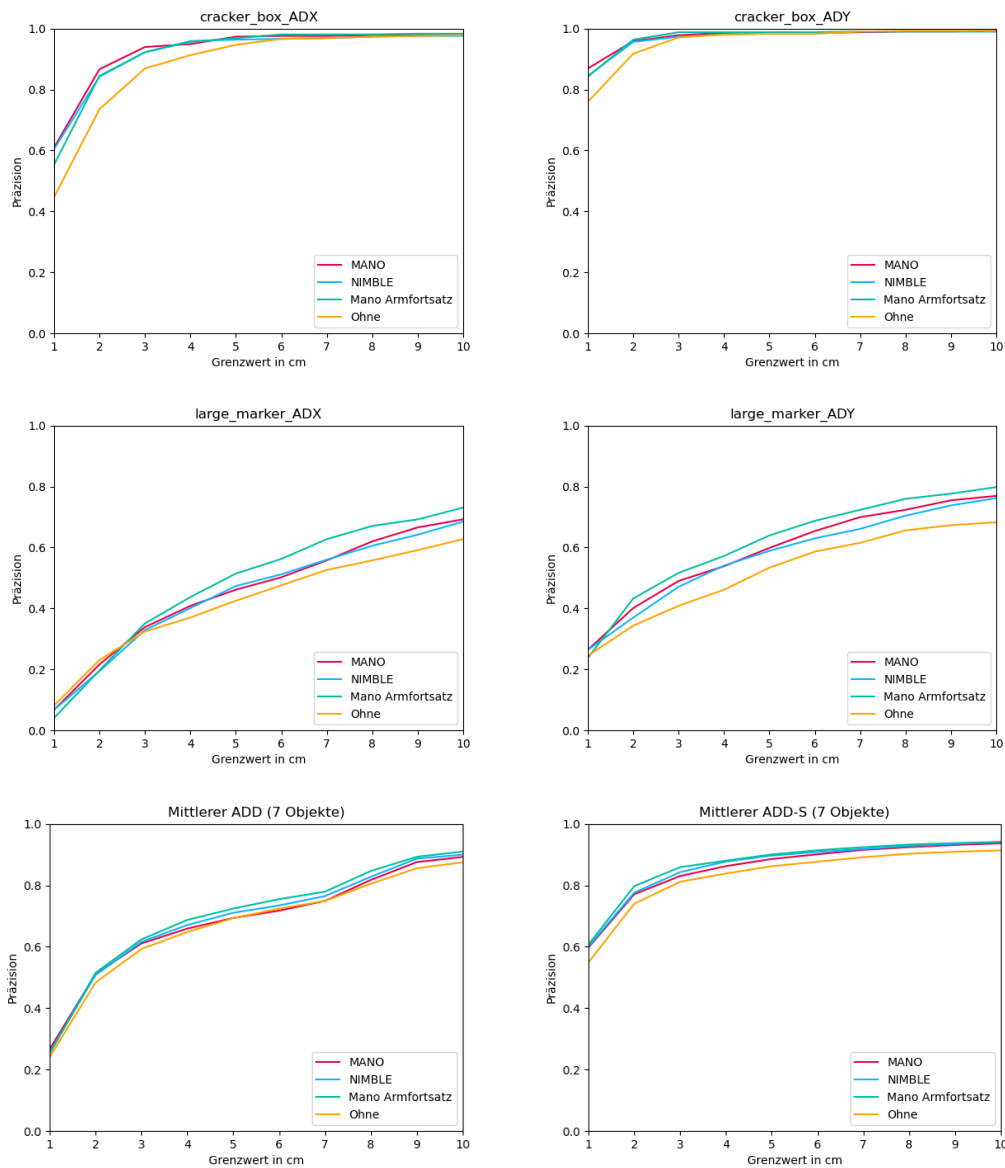


ABBILDUNG 4.2: Die Präzision/Grenzwert Kurven aus denen die AUC berechnet wird. Einmal für die cracker\_box oben, für den large\_marker mittig und über alle Objekte unten. Gezeigt sind alle vier Datensätze in einer gemeinsamen Grafik.

Abbildung 4.2 zeigt Näherungen der Kurven für die Berechnung der AUC. In diesem Fall für das durchschnittlich am besten (*cracker\_box*) und das am schlechtesten (*large\_marker*) erkannte Objekt. Für die *cracker\_box* ist lediglich erkennbar, dass

der Datensatz ohne Handinteraktion zu ungenaueren Vorhersagen führt. Die Objektpose für die `cracker_box` ist vergleichsweise einfach zu bestimmen, weshalb sich die Kurven ab einem Grenzwert von 6cm für ADD, respektive 3cm für ADD-S, bei sehr hoher Präzision von über 95% aneinander annähern. Bei den drei Datensätzen mit Handinteraktion lässt sich hier auch deshalb kein klarer Unterschied feststellen.

Die Kurven beim `large_marker` zeigen den Abfall des Datensatzes ohne Handinteraktion ebenfalls an, aufgrund der insgesamt schlechteren Erkennung zeigt sich hier allerdings die bereits zuvor beschriebene Tendenz, dass der Datensatz mit Armfortsatz leicht bessere Ergebnisse produziert als die MANO und NIMBLE Datensätze. An der Kurve zum ADD-S lässt sich hier sogar ein kleiner Vorsprung von MANO gegenüber NIMBLE erkennen.

Über alle Objekte mitteln sich diese Unterschiede beinahe heraus. In Anhang A.4 finden sich die Kurven für alle weiteren Objekte. Darin ist auch zu erkennen, dass der Datensatz, der zur besten Erkennungsfähigkeit führt, für verschiedene Objekte stark variiert. Es ist allerdings nach wie vor eine sehr leichte Tendenz zugunsten des Armfortsatz Datensatzes zu erkennen während sich NIMBLE und MANO kaum zu unterscheiden scheinen.

### 4.3 Gegriffene und nicht gegriffene Objekte

In Abbildung 4.1 ist zu erkennen, dass die Videosequenzen die Szene zu Beginn noch ohne Handinteraktion zeigen. Um hier eine differenziertere Analyse machen zu können, habe ich zwei weitere Experimente gemacht. Zum Einen habe ich die Testdaten aufgeteilt um eine differenziertere Analyse gegriffener und nicht gegriffener Objekte zu machen. Zum Anderen habe ich mit einem fünften Misch-Datensatz aus gleichem Anteil Szenen mit und ohne Handinteraktion trainiert.

#### 4.3.1 Teilung der Testdaten

Zum einen habe ich das Testset geteilt und grob in ein erstes Drittel ohne Handinteraktion (bis inklusive Frame 20) und die letzten zwei Drittel mit Handinteraktion aufgeteilt und die selben Netze erneut evaluiert. Repräsentativ für alle anderen Datensätze zeigt Tabelle 4.4 die Differenz der Ergebnisse mit dem MANO Datensatz zwischen dem ersten Drittel ohne Handinteraktion und den letzten beiden Dritteln mit Handinteraktion. Die Differenz zeigt klar, wie komplex die 6D Posenbestimmung in Hand-Objekt Interaktion im Vergleich zu frei liegenden Objekten ist. Ein Netz, das ausschließlich auf Bildern mit Hand-Objekt Interaktion trainiert wurde, schätzt Posen von freiliegenden Objekten trotzdem deutlich genauer als Posen von gegriffenen Objekten. Die Ergebnisse des Datensatzes ohne Handinteraktion fallen für die letzten beiden Drittel der Videosequenzen dementsprechend schlecht aus (siehe Anhang B.1/B.2). Zu bedenken ist hier allerdings, dass durch die Teilung der Testdaten das erste Drittel nur aus knapp mehr als 100 Testbilder besteht und durch die ungleiche Teilung die Testdatensätze unterschiedlich groß sind.

Objekt	ADD	ADD-S	AUC_ADD	AUC_ADD-S
cracker_box	-4.3	1.3	-4.2	-1.1
large_marker	-2.1	2.8	0.2	2.3
master_chef_can	-23.5	-7.2	-11.5	-4.5
mustard_bottle	-22.3	-13.9	-11.8	-10.5
potted_meat_can	-31.0	-20.9	-15.3	-8.2
sugar_box	-0.8	-5.3	-3.6	-5.2
tomato_soup_can	-10.2	-13.7	-6.0	-5.1
Mittel	-13.5	-8.1	-7.5	-4.6

TABELLE 4.4: Vergleich der Ergebnisse mit dem MANO Datensatz über alle Metriken. Die Einträge zeigen die Differenz (Frames ab Frame 20)-(Frames bis Frame 20). Überall dort, wo eine Metrik für ungegriffene Objekte besser ist, sind die Werte negativ und rot hervorgehoben. Grün für das Gegenteil und Schwarz für keinen Unterschied  $\pm$  Eins

### 4.3.2 50/50 Netz

Außerdem habe ich ein Netz auf einem Datensatz trainieren lassen, der zu 50% aus dem MANO Datensatz und zu den anderen 50% aus dem Datensatz ohne Handinteraktion besteht. Tabelle 4.5 zeigt eine geringe, aber sichtbare Verbesserung gegenüber dem reinen MANO Datensatz. Es lässt sich vermuten, dass für diesen Datensatz eine deutliche Verbesserung der Schätzung von ungegriffenen Objekten eine Verschlechterung der Erkennung von ungegriffenen Objekten maskieren könnte. Allerdings stützen die Ergebnisse des vollständigen Datensatzes ohne Handinteraktion bei der Evaluierung auf dem ersten Drittel der Videosequenzen diese Vermutung nicht, da sie sogar etwas schlechter ausfallen als für die Datensätze mit Handinteraktion (siehe Anhang B.1). Woher die beobachtete Verbesserung der Ergebnisse für diesen Datensatz herrührt bleibt zu untersuchen, da hier schließlich sogar nur auf halb so vielen Bildern mit sichtbarer Hand Interaktion trainiert wurde.

Objekt	MANO				MANO/Ohne			
	ADD	ADD-S	AUC	AUC-S	ADD	ADD-S	AUC	AUC-S
cracker_box	93.7	97.8	87.6	93.0	<b>93.9</b>	<b>98.1</b>	<b>87.9</b>	<b>93.5</b>
large_marker	7.9	<b>32.7</b>	<b>41.5</b>	<b>54.8</b>	<b>9.9</b>	29.3	41.0	52.0
master_chef_can	<b>47.9</b>	<b>88.8</b>	59.0	88.4	42.9	87.2	<b>63.3</b>	<b>88.6</b>
mustard_bottle	37.1	68.5	60.9	78.9	<b>40.8</b>	<b>74.6</b>	<b>64.6</b>	<b>81.6</b>
potted_meat_can	52.1	81.6	77.0	<b>88.9</b>	<b>52.6</b>	<b>84.2</b>	<b>78.7</b>	87.7
sugar_box	36.2	75.7	53.4	83.4	<b>37.8</b>	<b>80.6</b>	<b>58.6</b>	<b>86.7</b>
tomato_soup_can	28.9	64.2	65.5	79.4	<b>29.8</b>	<b>64.9</b>	<b>67.7</b>	<b>81.6</b>
Mittel	43.4	72.8	63.6	81.0	<b>43.9</b>	<b>74.1</b>	<b>66.0</b>	<b>81.7</b>

TABELLE 4.5: Vergleich des MANO Datensatzes mit dem geteilten Datensatz über alle Metriken. Pro Objekt ist jeweils der beste Wert einer Metrik über alle Datensätze **fett** markiert.

## Kapitel 5

# Fazit

### 5.1 Grenzen dieser Arbeit

Für meine synthetischen Datensätze habe ich nur sieben der 21 üblichen YCB Objekte genutzt. Es bleibt offen, ob eine breitere Objektauswahl aussagekräftigere Ergebnisse hervorgebracht hätte.

Ein großer Teil meiner Arbeit ist in die Optimierung und Entwicklung des Posen Editors geflossen. Während der Arbeit daran zeigten sich mit der Zeit einige Schwächen von Panda3D. Panda3D hat nur eine sehr limitierte Schnittstelle für das Auslesen und die Manipulation von einzelnen Vertices. Deshalb musste ich die Markierung von Vertices nach Blender auslagern. Außerdem war auch der Umfang an Features, die Panda3D bietet ein limitierender Faktor. Im Nachhinein wäre es möglicherweise besser gewesen, die initiale Hürde für die Integration der Modelle in eine ausgereifere Engine zu überwinden, um den Workflow für die Posenfindung zu vereinfachen.

Zum Ende der Arbeit habe ich gemerkt, wie durch den immer größer werdenden Anteil an manuellen Schritten in meiner Pipeline und der Näherung der Deadline die Nachvollziehbarkeit und Wissenschaftlichkeit in den Hintergrund gerückt ist. Ich hatte weniger Zeit um einige Kleinigkeiten zu überarbeiten, die den gesamten Prozess etwas stabiler gemacht hätten. Zum Beispiel fand die Randomisierung der Hand Formen effektiv nur vier Mal statt. Die allererste Hand ist mit einem Nullvektor für die Formparameter und der exakten Initialposition und -rotation erzeugt worden. Die Farben und Texturen sind von mir händisch ausgesucht worden, genau so wie die Kontaktvertices auf den Fingerspitzen und die Kontaktvertices auf den Objekten.

### 5.2 Fazit

Ich habe eine verlässliche Pipeline zur Erzeugung von synthetischen Daten für die 6D Posenbestimmung von gegriffenen Objekten erarbeitet. Diese Pipeline machte es mir möglich mehrere gleiche Datensätze mit unterschiedlichen Händen zu erstellen, um zu untersuchen welchen Einfluss diese Unterschiede auf die Erkennungsfähigkeit eines trainierten neuronalen Netzes haben.

Zuallererst konnte ich in meinen Experimenten bestätigen, dass die 6D Posenbestimmung von Objekten in Hand-Objekt Interaktionen ein komplexeres Problem darstellt als die Posenbestimmung freiliegender Objekte. Die Einbindung von Hand-Objekt Interaktionen in die Trainingsdaten führt hier zu einer sichtbaren Verbesserung gegenüber Trainingsdaten mit freiliegenden Objekten.

In meinen Experimenten konnte ich allerdings keinen erkennbaren Einfluss des Realismus von Handmodellen in synthetischen Trainingsdaten auf die Erkennungsfähigkeit eines Netzes feststellen. Angesichts der Tatsache, dass für das menschliche Auge zumindest ein visueller Unterschied der Hände zu erkennen ist, bleibt offen, ob dieser Unterschied für die Posenbestimmung von Objekten einfach nicht relevant ist oder ob die heutigen Netze vielleicht noch nicht komplex genug sind, als dass diese feinen Details im Verhältnis zu viel größeren Faktoren, wie dem Hintergrund, dem Licht und dem Level an Verdeckung einen nachweisbaren Unterschied machen.

Diese Ergebnisse bedeuten allerdings auch, dass für die Erzeugung synthetischer Datensätze mit Hand-Objekt Interaktion der Realismus des Handmodells nur eine untergeordnete oder überhaupt keine Rolle spielt. Im Falle von MANO und NIMBLE erreicht man durch den höheren Rechenaufwand des NIMBLE Modells keinerlei Verbesserung der Erkennungsfähigkeit eines Netzes, das MANO Modell wäre hier also völlig ausreichend.

Im Gegensatz dazu konnte ich für einige Objekte eine minimale Tendenz zur Verbesserung der Ergebnisse erzielen indem ich die MANO Hände durch eine sehr einfache Extrudierung des Handgelenks mit einem Armfortsatz erweitert habe. Angesichts dessen, dass in den Testdaten immer auch der Arm der greifenden Hand zu sehen ist, scheint eine akkurate Simulation des gesamten Armes intuitiv auch sinnvoll.

Die von mir beschriebenen Unterschiede in der Erkennungsfähigkeit sind allerdings überall so klein, dass sie statistisch kaum relevant sind. Vor allem in Anbetracht dessen, dass die Präzision der unterschiedlichen Netze sehr stark vom zu erkennenden Objekt abhängt, lässt sich aus meinen Ergebnissen nur schließen, dass die Verbesserung der Detailtreue von Händen für den hier untersuchten Anwendungszweck den Aufwand nicht wert ist.

### 5.3 Ausblick

Es bleibt zu untersuchen in wie weit die Simulation eines Armes oder weiterer Körperteile die leichte von mit beobachtete Tendenz verstärken könnten. Außerdem wäre es interessant zu überprüfen, welchen Einfluss die Varianz und der Realismus der Griffposen auf die Erkennungsfähigkeit für ein hier verwendetes Netz haben. Die Verdeckung, die durch die Hände entsteht, könnte der entscheidende Faktor bei der Posenbestimmung sein, womit sich die Frage ergibt, ob realistische Hände in diesem Fall möglicherweise sogar durch beliebige andere verdeckende Objekte ersetzt werden könnten. Des Weiteren bleibt noch zu untersuchen, ob die von mir erzeugten synthetischen Trainingsdaten zusammen mit realen Daten möglicherweise einen erkennbaren Mehrwert bieten.

Da ich bereits Handmodelle verwende und eine entsprechende Ground Truth exportiere, könnte man ebenfalls untersuchen ob sich die von mir generierten Daten auch für das Training eines Netzes zur Schätzung von Handposen eignen würden und wenn ja, welchen Einfluss die unterschiedlichen Handmodelle in diesem Fall hätten.

Zuletzt bietet auch die Pipeline Potenzial zum Ausbau. Vor im Hinblick auf die vielen manuellen Arbeitsschritte könnte die Generierung authentischer Griffposen weiter automatisiert werden.

## 5.4 Lessons Learned

### 5.4.1 Wissenschaftlichkeit

Aufgrund fehlender Erfahrung mit typischen Vorgängen großer wissenschaftlicher Arbeiten habe ich mich zu Beginn sehr stark auf die praktische Aufgabe verlassen, die Erzeugung eines Hand-Objekt Datensatzes. Erst im Laufe der Zeit habe ich mit meinem Betreuer eine wissenschaftliche Fragestellung herausgearbeitet, zu der ich Nachforschungen anstellen konnte. Erst bei der Recherche fiel mir auf, wie elementar frühzeitige, breite Recherche gewesen wäre, auch um eine interessante wissenschaftliche Fragestellung zu erarbeiten. Zu diesem Zeitpunkt hatte ich allerdings nicht unerheblich viel Zeit in die Entwicklung des Posen Editors gesteckt, so dass weniger Platz für die Einbindung aktueller wissenschaftlicher Erkenntnisse in dem Bereich war.

### 5.4.2 Ethik

Das Training und das Rendering benötigen sehr viel Speicherplatz und vor allem Zeit, die ein Rechner unter Vollast läuft. In wie weit der Energieverbrauch für den Einzelfall relevant ist und wie gerechtfertigt dieser Verbrauch für wissenschaftliche Zwecke ist, finde ich schwierig zu beurteilen. Klar ist aber, dass die Technologie der Neuronalen Netze üblicherweise deutlich rechenintensiver und speicherhungriger ist als klassische Algorithmen. Die damit einhergehende Steigerung des Energiebedarfs im Vergleich zu den meisten klassischen Methoden sollte beim Einsatz dieser Technologien immer gegen die Ergebnisse abgewogen werden. Zu wissenschaftlichen Zwecken lässt sich dieser Aufwand allerdings schwierig vermeiden. In meinem Fall fand das Training und Rendering auf Geräten in der Universität statt, die sich meines Wissens nach glücklicherweise um eine ökologische Stromversorgung bemüht, was den ökologischen Fußabdruck mildert und daher vor allem für energieintensive Anwendungen eigentlich eine Voraussetzung sein sollte.



## Anhang A

# Zusätzliche Bilder

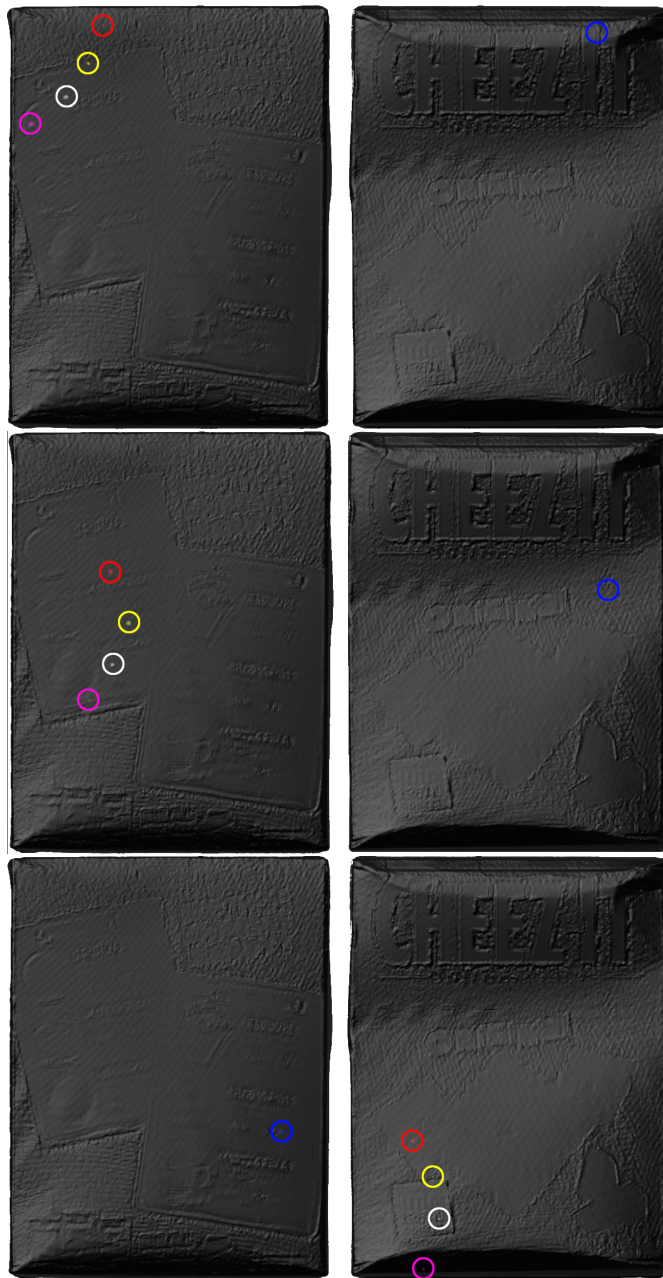


ABBILDUNG A.1: Die drei Kontaktpunktesets für das cracker\_box Objekt.



ABBILDUNG A.2: Beispielhaft 25 Bilder aus der gleichen Szene mit einer MANO Hand





ABBILDUNG A.3: Gute und schlechte geschätzte Posen visualisiert.  
Bilder aus dem DexYCB Datensatz [20]

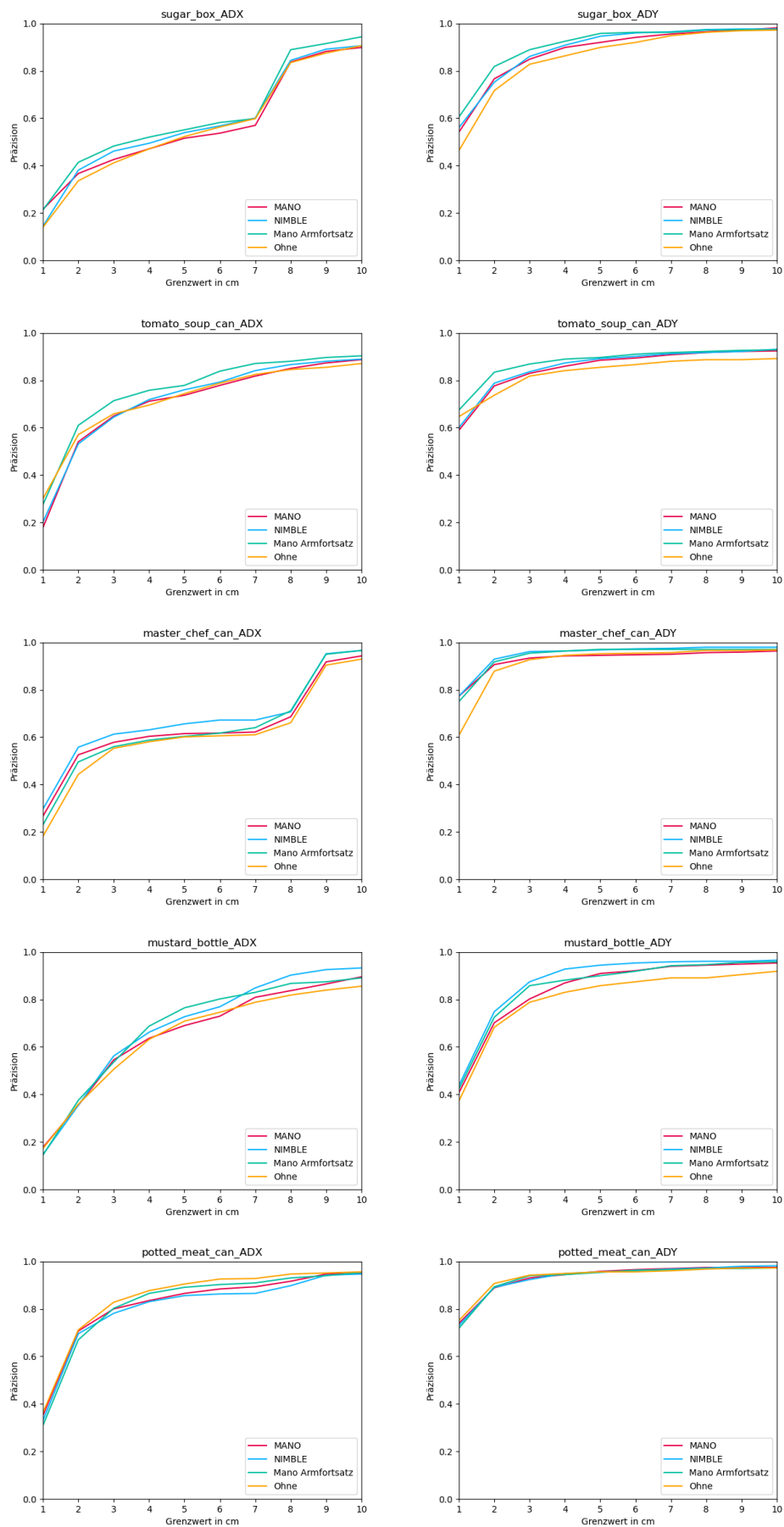


ABBILDUNG A.4: Grenzwert/Präzisions Kurven für die übrigen Objekte, die nicht aufgeführt wurden.

## Anhang B

# Vollständige Daten der Evaluation

Objekt	ADD	ADD-S	AUC_ADD	AUC_ADD-S
cracker_box	93.9	98.5	89.5	93.8
large_marker	6.8	35.6	52.1	65.9
master_chef_can	53.6	93.5	63.5	93.1
mustard_bottle	49.6	82.9	71.9	88.6
potted_meat_can	58.4	94.9	85.9	92.8
sugar_box	46.4	82.6	59.2	88.2
tomato_soup_can	38.2	72.8	73.3	86.4
Mittel	49.6	80.1	70.8	87.0
cracker_box	95.4	96.9	90.4	93.7
large_marker	9.8	30.3	42.3	54.6
master_chef_can	63.8	93.5	67.1	91.7
mustard_bottle	51.9	77.5	69.8	87.2
potted_meat_can	73.0	95.6	87.7	94.6
sugar_box	37.7	79.7	56.4	87.6
tomato_soup_can	36.8	74.3	70.4	84.2
Mittel	52.6	78.3	69.2	84.8
cracker_box	95.4	96.9	89.7	93.7
large_marker	15.9	35.6	49.2	60.9
master_chef_can	65.2	92.8	73.3	93.7
mustard_bottle	50.4	84.5	71.1	89.0
potted_meat_can	63.5	92.7	86.1	93.3
sugar_box	44.2	79.0	55.9	87.7
tomato_soup_can	21.3	61.8	68.6	82.7
Mittel	50.9	77.6	70.5	85.9
cracker_box	88.5	96.2	86.7	92.0
large_marker	18.9	37.9	49.3	62.2
master_chef_can	53.6	93.5	65.6	90.8
mustard_bottle	56.6	80.6	70.5	84.1
potted_meat_can	75.2	92.7	87.1	93.9
sugar_box	43.5	81.2	59.4	84.1
tomato_soup_can	46.3	72.8	71.4	83.2
Mittel	54.7	79.3	70.0	84.4

TABELLE B.1: 1. Drittel geteilter Datensatz

Objekt	ADD	ADD-S	AUC_ADD	AUC_ADD-S
cracker_box	87.9	97.9	85.4	92.6
large_marker	6.7	30.3	42.8	55.6
master_chef_can	41.9	84.9	57.5	88.0
mustard_bottle	31.0	65.3	60.4	76.7
potted_meat_can	41.0	75.8	72.9	86.4
sugar_box	39.6	81.4	56.4	85.3
tomato_soup_can	39.7	70.4	68.9	82.2
Mittel	41.1	72.3	63.5	81.0
cracker_box	91.1	98.2	86.2	92.6
large_marker	7.7	33.1	42.5	56.9
master_chef_can	40.3	86.2	55.6	87.2
mustard_bottle	29.7	63.7	58.0	76.7
potted_meat_can	42.0	74.7	72.3	86.4
sugar_box	36.8	74.4	52.8	82.3
tomato_soup_can	26.6	60.6	64.4	79.1
Mittel	39.2	70.1	61.7	80.2
cracker_box	87.9	97.2	85.3	92.2
large_marker	7.4	24.6	38.6	51.5
master_chef_can	43.6	87.6	58.1	88.7
mustard_bottle	30.0	69.3	61.2	80.6
potted_meat_can	38.6	74.1	70.8	86.0
sugar_box	34.7	74.0	53.1	82.0
tomato_soup_can	31.3	64.3	65.9	80.6
Mittel	39.1	70.2	61.9	80.2
cracker_box	80.1	95.7	80.8	90.1
large_marker	6.7	22.5	33.5	41.8
master_chef_can	31.2	81.5	52.2	85.7
mustard_bottle	25.7	61.3	56.2	73.5
potted_meat_can	42.3	76.8	75.7	86.9
sugar_box	27.4	66.3	49.8	79.5
tomato_soup_can	37.7	66.0	66.2	78.6
Mittel	35.9	67.2	59.2	76.6

TABELLE B.2: Die letzten beiden Drittel für den Testdatensatz. Für beide Tabellen gilt von oben nach unten: MANO mit Armfortsatz, MANO, NIMBLE, Ohne.

# Literatur

- [1] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. Adresse: [probml.ai](http://probml.ai).
- [2] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [3] T. Mitchell, *Machine Learning*. McGraw Hill, 1997.
- [4] Y. Lecun, C. Cortes und C. J.C. Burges, *THE MNIST DATABASE of handwritten digits*, <http://yann.lecun.com/exdb/mnist/>, Accessed: 2023-10-03.
- [5] S. Carter, Z. Armstrong, L. Schubert, I. Johnson und C. Olah, „Activation Atlas,“ *Distill*, 2019, <https://distill.pub/2019/activation-atlas>. DOI: [10.23915/distill.00015](https://doi.org/10.23915/distill.00015).
- [6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li und L. Fei-Fei, „ImageNet: A large-scale hierarchical image database,“ in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, S. 248–255. DOI: [10.1109/CVPR.2009.5206848](https://doi.org/10.1109/CVPR.2009.5206848).
- [7] O. Russakovsky, J. Deng, H. Su u. a., „ImageNet Large Scale Visual Recognition Challenge,“ *International Journal of Computer Vision*, Jg. 115, Nr. 3, S. 211–252, 2015, ISSN: 1573-1405. DOI: [10.1007/s11263-015-0816-y](https://doi.org/10.1007/s11263-015-0816-y). Adresse: <https://doi.org/10.1007/s11263-015-0816-y>.
- [8] J. Tremblay, A. Prakash, D. Acuna u. a., *Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization*, 2018. arXiv: [1804.06516](https://arxiv.org/abs/1804.06516) [cs.CV].
- [9] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba und P. Abbeel, *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*, 2017. arXiv: [1703.06907](https://arxiv.org/abs/1703.06907) [cs.R0].
- [10] K. Bousmalis, A. Irpan, P. Wohlhart u. a., *Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping*, 2017. arXiv: [1709.07857](https://arxiv.org/abs/1709.07857) [cs.LG].
- [11] M. Rudorfer, L. Neumann und J. Kruger, „Towards Learning 3d Object Detection and 6d Pose Estimation from Synthetic Data,“ en, in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain: IEEE, Sep. 2019, S. 1540–1543, ISBN: 978-1-72810-303-7. DOI: [10.1109/ETFA.2019.8869318](https://doi.org/10.1109/ETFA.2019.8869318). Adresse: <https://ieeexplore.ieee.org/document/8869318/> (besucht am 28. 08. 2023).
- [12] F. Mueller, F. Bernard, O. Sotnychenko u. a., „GANerated hands for real-time 3d hand tracking from monocular RGB,“ in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT: IEEE, Juni 2018, S. 49–59, ISBN: 978-1-5386-6420-9. DOI: [10.1109/CVPR.2018.00013](https://doi.org/10.1109/CVPR.2018.00013). Adresse: <https://ieeexplore.ieee.org/document/8578111/> (besucht am 14. 09. 2023).



- [13] R. Duda. „Physically Based Rendering: Viel Realismus, aber bitte mit kreativen Stellschrauben!“ (Mai 2019), Adresse: <https://www.digitalproduction.com/2019/05/25/physically-based-rendering-viel-realismus-aber-bitte-mit-kreativen-stellschrauben/>.
- [14] Z. He, W. Feng, X. Zhao und Y. Lv, „6d pose estimation of objects: Recent technologies and challenges,“ *Applied Sciences*, Jg. 11, Nr. 1, S. 228, 29. Dez. 2020, ISSN: 2076-3417. DOI: [10.3390/app11010228](https://doi.org/10.3390/app11010228). Adresse: <https://www.mdpi.com/2076-3417/11/1/228> (besucht am 02. 10. 2023).
- [15] M. Sundermeyer, T. Hodan, Y. Labbe u. a., *BOP challenge 2022 on detection, segmentation and pose estimation of specific rigid objects*, 25. Feb. 2023. arXiv: [2302.13075](https://arxiv.org/abs/2302.13075)[cs]. Adresse: <http://arxiv.org/abs/2302.13075> (besucht am 28. 08. 2023).
- [16] C. Sahin, G. Garcia-Hernando, J. Sock und T.-K. Kim, *A review on object pose recovery: From 3d bounding box detectors to full 6d pose estimators*, 19. Apr. 2020. arXiv: [2001.10609](https://arxiv.org/abs/2001.10609)[cs]. Adresse: <http://arxiv.org/abs/2001.10609> (besucht am 28. 08. 2023).
- [17] Y. Xiang, T. Schmidt, V. Narayanan und D. Fox, *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*, en, arXiv:1711.00199 [cs], Mai 2018. Adresse: <http://arxiv.org/abs/1711.00199> (besucht am 27. 07. 2023).
- [18] B. Calli, A. Singh, J. Bruce u. a., „Yale-CMU-Berkeley dataset for robotic manipulation research,“ *The International Journal of Robotics Research*, Jg. 36, Nr. 3, S. 261–268, 2017. DOI: [10.1177/0278364917700714](https://doi.org/10.1177/0278364917700714). eprint: <https://doi.org/10.1177/0278364917700714>. Adresse: <https://doi.org/10.1177/0278364917700714>.
- [19] T. Hodan, M. Sundermeyer, B. Drost u. a., *BOP Challenge 2020 on 6D Object Localization*, 2020. arXiv: [2009.07378](https://arxiv.org/abs/2009.07378) [cs.CV].
- [20] Y.-W. Chao, W. Yang, Y. Xiang u. a., „DexYCB: A Benchmark for Capturing Hand Grasping of Objects,“ in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [21] S. Brahmabhatt, C. Tang, C. D. Twigg, C. C. Kemp und J. Hays, *ContactPose: A dataset of grasps with object contact and hand pose*, 18. Juli 2020. arXiv: [2007.09545](https://arxiv.org/abs/2007.09545)[cs]. Adresse: <http://arxiv.org/abs/2007.09545> (besucht am 15. 09. 2023).
- [22] S. Sridhar, F. Mueller, M. Zollhöfer, D. Casas, A. Oulasvirta und C. Theobalt, „Real-time joint tracking of a hand manipulating an object from RGB-d input,“ in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe und M. Welling, Hrsg., Bd. 9906, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, S. 294–310, ISBN: 978-3-319-46474-9 978-3-319-46475-6. DOI: [10.1007/978-3-319-46475-6\\_19](https://doi.org/10.1007/978-3-319-46475-6_19). Adresse: [http://link.springer.com/10.1007/978-3-319-46475-6\\_19](http://link.springer.com/10.1007/978-3-319-46475-6_19) (besucht am 15. 09. 2023).
- [23] Z. Cao, I. Radosavovic, A. Kanazawa und J. Malik, „Reconstructing hand-object interactions in the wild,“ in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Okt. 2021, S. 12397–12406, ISBN: 978-1-66542-812-5. DOI: [10.1109/ICCV48922.2021.01219](https://doi.org/10.1109/ICCV48922.2021.01219). Adresse: <https://ieeexplore.ieee.org/document/9710775/> (besucht am 14. 09. 2023).

- [24] Z. Fan, O. Taheri, D. Tzionas u. a., „ARCTIC: A Dataset for Dexterous Bimanual Hand-Object Manipulation,“ in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
- [25] G. Garcia-Hernando, S. Yuan, S. Baek und T.-K. Kim, *First-person hand action benchmark with RGB-d videos and 3d hand pose annotations*, 10. Apr. 2018. arXiv: [1704.02463\[cs\]](https://arxiv.org/abs/1704.02463). Adresse: <http://arxiv.org/abs/1704.02463> (besucht am 15.09.2023).
- [26] S. Hampali, M. Rad, M. Oberweger und V. Lepetit, „HONotate: A method for 3D Annotation of Hand and Object Poses,“ 2020.
- [27] Y. Hasson, G. Varol, D. Tzionas u. a., *Learning joint reconstruction of hands and manipulated objects*, 11. Apr. 2019. arXiv: [1904.05767\[cs\]](https://arxiv.org/abs/1904.05767). Adresse: <http://arxiv.org/abs/1904.05767> (besucht am 14.09.2023).
- [28] E. Corona, A. Pumarola, G. Alenya, F. Moreno-Noguer und G. Rogez, „Gan-Hand: Predicting human grasp affordances in multi-object scenes,“ in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Juni 2020, S. 5030–5040, ISBN: 978-1-72817-168-5. DOI: [10.1109/CVPR42600.2020.00508](https://doi.org/10.1109/CVPR42600.2020.00508). Adresse: <https://ieeexplore.ieee.org/document/9156512/> (besucht am 14.09.2023).
- [29] N. Wheatland, Y. Wang, H. Song, M. Neff, V. Zordan und S. Jörg, „State of the art in hand and finger modeling and animation,“ *Computer Graphics Forum*, Jg. 34, Nr. 2, S. 735–760, Mai 2015, ISSN: 01677055. DOI: [10.1111/cgf.12595](https://doi.org/10.1111/cgf.12595). Adresse: <https://onlinelibrary.wiley.com/doi/10.1111/cgf.12595> (besucht am 15.09.2023).
- [30] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle und X. Twombly, „Vision-based hand pose estimation: A review,“ en, *Computer Vision and Image Understanding*, Jg. 108, Nr. 1-2, S. 52–73, Okt. 2007, ISSN: 10773142. DOI: [10.1016/j.cviu.2006.10.012](https://doi.org/10.1016/j.cviu.2006.10.012). Adresse: <https://linkinghub.elsevier.com/retrieve/pii/S1077314206002281> (besucht am 28.08.2023).
- [31] H. Cheng, L. Yang und Z. Liu, „Survey on 3D Hand Gesture Recognition,“ en, *IEEE Transactions on Circuits and Systems for Video Technology*, Jg. 26, Nr. 9, S. 1659–1673, Sep. 2016, ISSN: 1051-8215, 1558-2205. DOI: [10.1109/TCSVT.2015.2469551](https://doi.org/10.1109/TCSVT.2015.2469551). Adresse: <http://ieeexplore.ieee.org/document/7208833/> (besucht am 28.08.2023).
- [32] W. Zhao, J. Chai und Y.-Q. Xu, „Combining marker-based mocap and RGB-d camera for acquiring high-fidelity hand motion data,“ *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 10 pages, 2012, Artwork Size: 10 pages ISBN: 9783905674378 Publisher: The Eurographics Association, ISSN: 1727-5288. DOI: [10.2312/SCA/SCA12/033-042](https://doi.org/10.2312/SCA/SCA12/033-042). Adresse: <http://diglib.eg.org/handle/10.2312/SCA.SCA12.033-042> (besucht am 15.09.2023).
- [33] A Hollister, W. L. Buford, L. M. Myers, D. J. Giurintano und A Novick, „The axes of rotation of the thumb carpometacarpal joint,“ en, *J. Orthop. Res.*, Jg. 10, Nr. 3, S. 454–460, Mai 1992.
- [34] I. Oikonomidis, N. Kyriazis und A. Argyros, „Efficient model-based 3d tracking of hand articulations using kinect,“ in *Proceedings of the British Machine Vision Conference 2011*, Dundee: British Machine Vision Association, 2011, S. 101.1–101.11, ISBN: 978-1-901725-43-8. DOI: [10.5244/C.25.101](https://doi.org/10.5244/C.25.101). Adresse: <http://www.bmva.org/bmvc/2011/proceedings/paper101/index.html> (besucht am 18.09.2023).

- [35] T. Schmidt, R. Newcombe und D. Fox, „DART: Dense articulated real-time tracking,“ in *Robotics: Science and Systems X*, Robotics: Science und Systems Foundation, 12. Juli 2014, ISBN: 978-0-9923747-0-9. DOI: [10.15607/RSS.2014.X.030](https://doi.org/10.15607/RSS.2014.X.030). Adresse: <http://www.roboticsproceedings.org/rss10/p30.pdf> (besucht am 18.09.2023).
- [36] A. Tkach, M. Pauly und A. Tagliasacchi, „Sphere-meshes for real-time hand modeling and tracking,“ *ACM Transactions on Graphics*, Jg. 35, Nr. 6, S. 1–11, 11. Nov. 2016, ISSN: 0730-0301, 1557-7368. DOI: [10.1145/2980179.2980226](https://doi.org/10.1145/2980179.2980226). Adresse: <https://dl.acm.org/doi/10.1145/2980179.2980226> (besucht am 18.09.2023).
- [37] S. Melax, L. Keselman und S. Orsten, „Dynamics based 3d skeletal hand tracking,“ in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Orlando Florida: ACM, 21. März 2013, S. 184–184, ISBN: 978-1-4503-1956-0. DOI: [10.1145/2448196.2448232](https://doi.org/10.1145/2448196.2448232). Adresse: <https://dl.acm.org/doi/10.1145/2448196.2448232> (besucht am 18.09.2023).
- [38] S. Khamis, J. Taylor, J. Shotton, C. Keskin, S. Izadi und A. Fitzgibbon, „Learning an efficient model of hand shape variation from depth images,“ in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Juni 2015, S. 2540–2548, ISBN: 978-1-4673-6964-0. DOI: [10.1109/CVPR.2015.7298869](https://doi.org/10.1109/CVPR.2015.7298869). Adresse: <http://ieeexplore.ieee.org/document/7298869/> (besucht am 18.09.2023).
- [39] J. Romero, D. Tzionas und M. J. Black, „Embodied Hands: Modeling and Capturing Hands and Bodies Together,“ *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 245:1–245:17, Jg. 36, Nr. 6, Nov. 2017.
- [40] G. Moon, T. Shiratori und K. M. Lee, *DeepHandMesh: A weakly-supervised deep encoder-decoder framework for high-fidelity hand mesh modeling*, 18. Aug. 2020. arXiv: [2008.08213\[cs\]](https://arxiv.org/abs/2008.08213). Adresse: <http://arxiv.org/abs/2008.08213> (besucht am 18.09.2023).
- [41] Y. Li, M. Wu, Y. Zhang, L. Xu und J. Yu, *PIANO: A parametric hand bone model from magnetic resonance imaging*, 21. Juni 2021. arXiv: [2106.10893\[cs\]](https://arxiv.org/abs/2106.10893). Adresse: <http://arxiv.org/abs/2106.10893> (besucht am 18.09.2023).
- [42] Y. Li, L. Zhang, Z. Qiu u. a., „NIMBLE: A Non-Rigid Hand Model with Bones and Muscles,“ *ACM Trans. Graph.*, Jg. 41, Nr. 4, 2022, ISSN: 0730-0301. DOI: [10.1145/3528223.3530079](https://doi.org/10.1145/3528223.3530079). Adresse: <https://doi.org/10.1145/3528223.3530079>.
- [43] A. Miller und P. Allen, „Graspit! A versatile simulator for robotic grasping,“ *IEEE Robotics and Automation Magazine*, Jg. 11, Nr. 4, S. 110–122, 2004. DOI: [10.1109/MRA.2004.1371616](https://doi.org/10.1109/MRA.2004.1371616). Adresse: <http://graspit-simulator.github.io/>.
- [44] A. S. Lakshmipathy, N. Feng, Y. X. Lee, M. Mahler und N. S. Pollard, *Contact edit: Artist tools for intuitive modeling of hand-object interactions*, 18. Mai 2023. arXiv: [2305.02051\[cs\]](https://arxiv.org/abs/2305.02051). Adresse: <http://arxiv.org/abs/2305.02051> (besucht am 28.08.2023).
- [45] R. Parent, *Computer Animation: Algorithms and Techniques, Third Edition*. Morgan Kaufmann, 2012, <https://learning.oreilly.com/library/view/computer-animation-3rd/9780124158429/> (visited 2023-09-15).

- [46] A. Aristidou, J. Lasenby, Y. Chrysanthou und A. Shamir, „Inverse Kinematics Techniques in Computer Graphics: A Survey,“ *Computer Graphics Forum*, Jg. 37, Nr. 6, S. 35–58, 2018. DOI: <https://doi.org/10.1111/cgf.13310>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13310>. Adresse: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13310>.
- [47] T. Hodaň, F. Michel, E. Brachmann u. a., „BOP: Benchmark for 6d object pose estimation,“ in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu und Y. Weiss, Hrsg., Bd. 11214, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2018, S. 19–35, ISBN: 978-3-030-01248-9 978-3-030-01249-6. DOI: [10.1007/978-3-030-01249-6\\_2](https://doi.org/10.1007/978-3-030-01249-6_2). Adresse: [https://link.springer.com/10.1007/978-3-030-01249-6\\_2](https://link.springer.com/10.1007/978-3-030-01249-6_2) (besucht am 28.08.2023).
- [48] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel und A. M. Dollar, „Benchmarking in manipulation research: Using the yale-CMU-berkeley object and model set,“ *IEEE Robotics & Automation Magazine*, Jg. 22, Nr. 3, S. 36–52, Sep. 2015, ISSN: 1070-9932. DOI: [10.1109/MRA.2015.2448951](https://doi.org/10.1109/MRA.2015.2448951). Adresse: <http://ieeexplore.ieee.org/document/7254318/> (besucht am 20.09.2023).
- [49] M. Davis und N. Holbrook. „Unicode Technical Standard 51.“ (31. Aug. 2022), Adresse: <http://www.unicode.org/reports/tr51>.
- [50] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll und M. J. Black, „SMPL: A skinned multi-person linear model,“
- [51] Y. Zhou, *Minimal-IK*, <https://github.com/CalciferZh/Minimal-IK>, 2020.
- [52] Carnegie Mellon University, *Panda3D 1.10.13*. Adresse: <https://www.panda3d.org/>.
- [53] O. Taheri, *MANO*, <https://github.com/otaheri/MANO>, 2020.
- [54] M. Denninger, D. Winkelbauer, M. Sundermeyer u. a., „BlenderProc2: A Procedural Pipeline for Photorealistic Rendering,“ *Journal of Open Source Software*, Jg. 8, Nr. 82, S. 4901, 2023. DOI: [10.21105/joss.04901](https://doi.org/10.21105/joss.04901). Adresse: <https://doi.org/10.21105/joss.04901>.
- [55] Y. Su, M. Saleh, T. Fetzler u. a., „ZebraPose: Coarse to Fine Surface Encoding for 6DoF Object Pose Estimation,“ *arXiv preprint arXiv:2203.09418*, 2022.