Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Hannes Dieck

Modellbasierte Handposenerkennung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hannes Dieck

Modellbasierte Handposenerkennung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Birgit Wendholt
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 26. Oktober 2013

Hannes Dieck

Thema der Arbeit
Modellbasierte Handposenerkennung

Stichworte
Mensch-Computer-Interaktion, Bildverarbeitung, Hand, Posenerkennung, Handposenerkennung, modell-basiert, Tracking, Gesten, Kinect

Kurzzusammenfassung
Die Tendenz zur zunehmenden Informatisierung des menschlichen Alltags dient häufig als Motivation für Arbeiten aus dem Forschungsbereich Human Computer Interaction. So sind in naher Zukunft intuitive natürlich bedienbare Benutzerschnittstellen erforderlich, um die Interaktion mit Geräten aber auch den allgemeinen Umgang mit den immer bildlicher und immersiver werdenden Informationen so komfortabel wie möglich zu gestalten. Dabei stellt die Hand und deren einzelne Bewegungsgrade das wohl natürlichste Werkzeug des Menschen zur allgemeinen Interaktion mit seiner Umgebung dar. Unter dieser Prämisse soll in dieser Bachelorarbeit eine visuelle modellbasierte Handposenerkennung realisiert und in Form eines Framework zur allgemeinen Nutzung bereitgestellt werden. Die Lösung erfordert dabei eine vorangehende Erstellung eines parametrisierten Handmodells, mit dessen Hilfe sich, basierend auf dem Konzept einer Partikel Schwarm Optimierung (PSO), eine kontinuierliche Erkennung von Handposen erreichen lässt. Zur Beschleunigung des Erkennungsprozesses werden zudem die technischen Eigenschaften eines konventionellen Grafikprozessors ausgenutzt.
Hannes Dieck

Title of the paper
Model-Based Hand Pose Recognition

Keywords
HCI, human computer interaction, computer vision, Hand, pose estimation, hand pose estimation, model-based, tracking gesture recognition, kinect

Abstract
Research in field of Human Computer Interaction is still yet motiviated by the increasing trend towards ubiquitous computing and dissemination of informations. This seems reasonable, since there is a demand for natural user interfaces, which helps to controll devices, but also makes dealing with general informational content, which becomes increasingly visual and immersive, as comfortable as possible. Thereby, the hand and it's degrees of freedom propably represents the most natural tool for human interaction. Based on this assumption, this bachelor thesis

focusses on development of a model-based pose estimation framework, which can serve as general tool for common use. So the solution makes a description of a parametric hand model necessary. Based on this hand model and the conceptual support of Particle Swarm Optimization, a continous hand pose estimation can be achieved. To speed up recognition process, this solution additionally utilizes technical capabilities of an common Graphics Processing Unit.

Inhaltsverzeichnis

1 Einleitung

Mit der zunehmenden Informatisierung der Gesellschaft hält auch der Computer immer häufiger Einzug in den menschlichen Alltag. Die eigenen Vier Wände ohne intelligenten TV, der dem Benutzer neben dem reinen Fernsehempfang eine Reihe weiterer Anwendungen offenbart? Heutzutage kaum noch vorstellbar. Mobiltelefone, die neben der mehrkanaligen Kommunikation als handliches universelles Endgerät den alltäglichen Umgang mit multimedialen Inhalten erlauben, sind ebenfalls keine Besonderheit mehr. Mit dieser, als Ubiquitous Computing bezeichneten, Ausbreitung der informationsverarbeitenden Technologie in sämtliche Lebenslagen des Menschen, gewinnt die Frage der Interaktion zwischen Mensch und Computer zunehmend an Bedeutung. Die traditionelle Bedienung über Maus und Tastatur erscheint dabei, je nach Gerät und Anwendung, an ihre Grenzen zu stoßen. So ist die Suche nach alternativen natürlicheren Interaktionsformen zwischen Mensch und Computer bereits Thema vieler Arbeiten aus dem Bereich der *Human Computer Interaction* (HCI). Die Hand als universelles Werkzeug erscheint dabei als besonders geeignet, um den Menschen eine intuitive natürliche Interaktion zu ermöglichen. Doch wie erkennt der Computer eine Hand?

"*Finding a hand-detector certainly did not allow us to program one.*"
David Marr (1945-1980)

1.1 Motivation

Eine der intuitivsten natürlichsten Möglichkeiten des Menschen mit seiner Umwelt zu interagieren liegt im Gebrauch der Hände. So kann die Hand genutzt werden, um Informationen aus der Umgebung aufzunehmen, aktiv mit ihr zu interagieren, einzelne Objekte als Werkzeuge zu nutzen oder Informationen an die Umgebung zu vermitteln. Die Einsatzmöglichkeiten der Hand scheinen nahezu unbegrenzt. Ein Aspekt, der in der Forschung aber auch der Industrie häufig das Interesse weckt, die Hand als artikulierten Informationsträger wahrzunehmen, um die gelieferten Informationen der Hand schließlich je nach Kontext interpretieren zu können.

Zu den klassischen Anwendungsgebieten, die sich der Hand-basierten Interaktion bedienen, zählt die Erkennung von Gesten und Gebärden oder die Fernsteuerung von Anwendungen. Diese meist rein symbolische Form der Interaktion erweist sich heutzutage in vielen HCI-Szenarien als nicht mehr ausreichend. So besteht ein vermehrtes Interesse an Anwendungen aus unterschiedlichsten Bereichen, um das Artikulationsvermögen der Hand intensiver in ihrem Bedie-

nungskonzept zu berücksichtigen. Dies reicht bis zu Lösungen aus dem Bereich der *Grasping Interaction* (s. Abb. 2.3), die den natürlichen Umgang der Hand mit Objekten, im Sinne von Greifen und Anfassen, in einem bestimmten Umfeld wahrnehmen und indirekt auf ein Steuerkonzept abbilden kann. Als Beispiel dafür können Anwendungen aus dem Bereich Virtueller- oder Erweiterter-Realität sowie *Games* gelten, die die reale Hand als ultimativen Controller zur Manipulation virtueller Objekte in einer immersiven Umgebung betrachten (Zhu u. a., 2006).

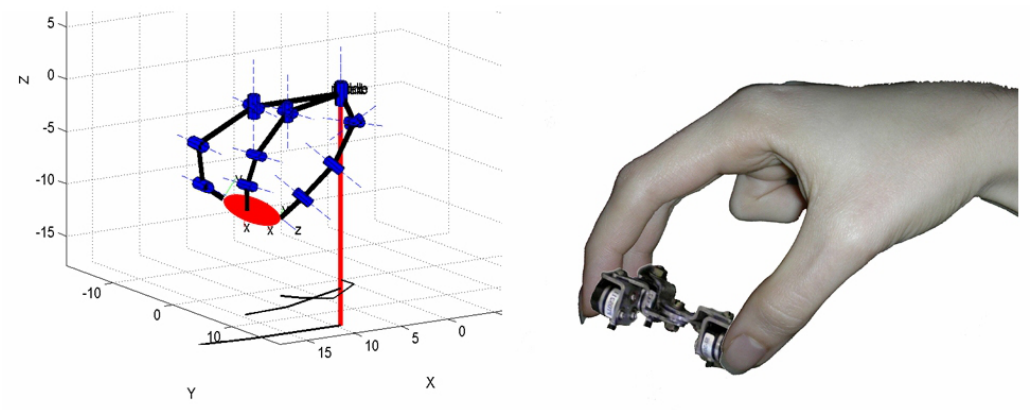


Abbildung 1.1: Grasping Interaction. Quelle: sirslab.dii.unisi.it (2013)

Die erweiterte Interaktion zwischen Mensch und Anwendung durch die Hand erfordert dabei eine konzeptionelle Lösung zur Erkennung der Hand, die meist über die Fähigkeiten der klassischen statischen oder dynamischen Gestenerkennung hinaus gehen und so zum Thema der Handposenerkennung führen.

Ein häufig genanntes Ziel ist dabei die Realisierung einer markerlosen visuellen Handposenerkennung, da diese keine tragbaren Geräte oder Objekte zur Wahrnehmung der Handpose voraussetzt und eine flexiblere natürlichere Interaktion möglich wird. Die visuelle Handposenerkennung stellt jedoch eine ganz eigene Herausforderung dar. So besteht ein wesentliches Problem darin, dass die Hand als artikuliertes Objekt mit ihrer hohen Anzahl an Bewegungs-Freiheitsgraden auch eine nahezu unbegrenzte Anzahl an möglichen Darstellungen im Bild einer Kamera ergibt. Um die Handpose über die Menge aller möglichen Darstellungen hinaus generell Erkennen zu können, bedarf es besonderen Lösungsstrategien. Ein etablierter Ansatz aus dem Bereich der Objekterkennung zur Lösung eines solches Problems liegt in der Zuhilfenahme eines Modells. So bietet sich auch für die Realisierung einer Handposenerkennung ein modellbasierter Ansatz an, bei dem ein dynamisches Handmodell kontinuierlich so lange angepasst wird, bis dessen Darstellung mit der einer observierten Hand übereinstimmt und das Modell die wahrgenommene Handpose reflektiert.

1.2 Zielsetzung

Im Rahmen dieser Bachelorarbeit soll eine visuelle modellbasierte Handposenerkennung entwickelt werden. Dazu soll eine Erstellung eines parametrisierten Modells erfolgen, mit dessen Hilfe die aktuelle Pose einer, in einem Eingabebild dargestellten, Hand ermittelt werden kann. Die für die Erkennung notwendige Funktionalität soll in einem Framework gekapselt zur einfachen Verwendung bereitstehen. Um die Geschwindigkeit der Erkennung zu erhöhen, soll diese partiell von den Fähigkeiten zur parallelisierten Verarbeitung eines Grafikprozessors Gebrauch machen.

1.3 Gliederung

In Kapitel 2 erfolgt eine intensive Analyse und Bewertung der vergleichbaren Arbeiten. Aus den daraus gewonnenen Erkenntnisse lassen sich die, für die Posenerkennung notwendigen, Schritte und verwendbaren Algorithmen ableiten.
Die Realisierung der Posenerkennung erfolgt im Rahmen von Kapitel 3. Darin wird zwischen einer im Abschnitt 3.2 beschriebenen *offline*-Phase, die die Erstellung der Modelle umfasst, und einer *online*-Phase (s. Abschnitt 3.3), in der die eigentliche Definition der Erkennungsfunktionalität erfolgt, unterschieden. Zusätzlich werden im Kapitel 3 die allgemeinen Voraussetzungen für den Betrieb der Posenerkennung angegeben und eine Hardware-Auswahl getroffen.
In Kapitel 4 erfolgt schließlich eine allgemeine Zusammenfassung und Bewertung der Arbeit mit einem finalen Ausblick auf zu erwartende Trends im Bereich der Handposenerkennung.

2 Vergleichbare Arbeiten

Über die Jahre wurde bereits viel Forschungsarbeit zum Thema visuelle Handposenerkennung geleistet. Dies lässt sich auch an der hohen Anzahl veröffentlichter Literatur ablesen. Dabei kann die Handposenerkennung als Teilbereich der Körperposen- oder der allgemeinen Erkennung artikulierter Objekte aufgefasst werden. Die verwendeten Verfahren sind daher häufig aus diesen Schwerpunktthemen entliehen. Erol u. a. (2007) konzentrieren sich in ihrer Zusammenfassung auf die visuelle Erkennung von Handposen und liefern damit einen ersten Ansatz für diese Arbeit.

Dabei unterscheiden sie zwischen Lösungen, die eine partielle Erkennung der Handpose realisieren (*Partial Hand Pose Estimation*), und Verfahren, die eine generelle Erkennung der kompletten Pose zum Ziel haben (*Full DOF Hand Pose Estimation*). Die partielle Erkennung bedeutet dabei, dass nur die Lage einzelner Elemente der Hand (z. B. Finger) oder nur eine bestimmte Menge an vorab definierten Posen bestimmt werden.

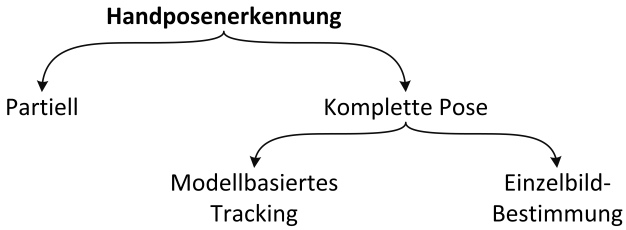


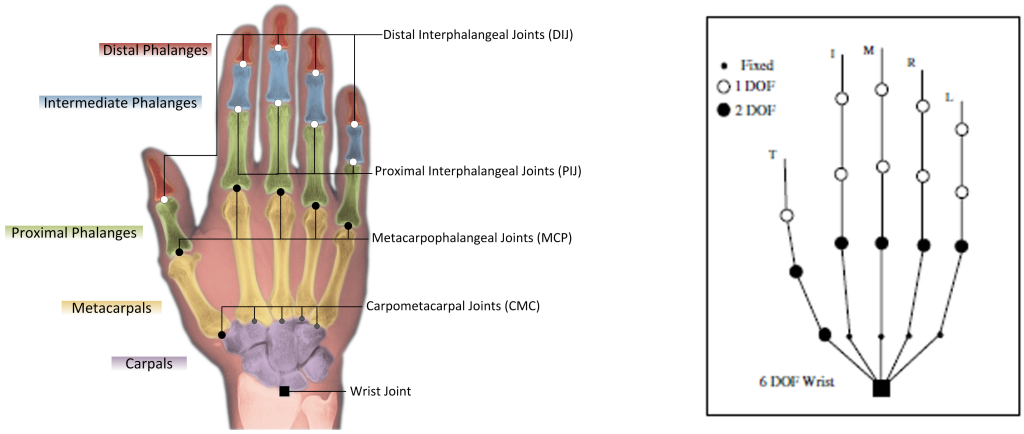
Abbildung 2.1: Kategorien von Handposenerkennungen (i.A.a. Erol u. a., 2007)

Für die generelle Erkennung der kompletten Handpose im Bild der Kamera, haben sich dagegen zwei unterscheidbare Ansätze etabliert. Das *Model-based Pose Tracking* und die *Single View Pose Estimation* (s. Abb. 2.1). Beim modellbasiertem Ansatz wird ein Handmodell über die Zeit an die kontinuierliche Aufnahme der Kamera angepasst und reflektiert letztendlich die eingenommene Pose. Die *Single View Estimation* bedeutet dagegen die Handpose direkt aus einem Eingabebild ohne den Umweg über ein Modell zu erkennen.

In diesem Kapitel soll ein Überblick über den Stand der Technik im Bereich der visuellen Handposenerkennung gegeben werden. Zur Einführung in die Thematik erfolgt im nächsten Abschnitt 2.1 eine Schilderung des allgemeinen Problembereichs der visuellen Handposenerkennung. Im darauf folgenden Abschnitt 2.2 wird der Schritt einer Vorverarbeitung im Sinne einer Segmentierung und Merkmalsextraktion diskutiert. Im Abschnitt 2.3 werden dann partielle Verfahren vorgestellt. Schließlich wird auf die Erkennung kompletter Handposen eingegangen indem Lösungen im Bereich der *Single View Pose Estimation* (s. Abschn. 2.4) und des *Model-based Pose Tracking* (s. Abschn. 2.5) genauer analysiert werden.

2.1 Allgemeiner Problembereich der visuellen Handposenerkennung

Das primäre Ziel einer visuellen Erkennung von Handposen aus dem Bild einer oder mehrerer Kameras liegt darin, aus der pixelbasierten Repräsentation des Eingabebildes, die keine Zusammenhänge zwischen einzelnen Objekten beschreibt, die Hand und deren eingenommene Pose abzuleiten. Erschwerend wirkt hierbei, dass im Gegensatz zu Objekten mit einer festen Morphologie die Hand als artikuliertes Objekt mit einer sehr hohen Beweglichkeit über die Zeit beliebig viele Posen einnehmen kann. Dies führt zu einer sehr hohen Anzahl möglicher Darstellungen, was eine generelle Erkennung der Handpose aus einem Eingabebild erschwert. Die Beweglichkeit der Hand lässt sich durch die einzelnen Bewegungsmöglichkeiten an den Gelenken bzw. den Verbindungspunkten zwischen den einzelnen Knochen (Joints) der Hand beschreiben (s. Abb. 2.2). Diese kann man als mehr oder wenig eingeschränkte Rotationen um die einzelnen Verbindungspunkt auffassen. Als Synonym für die einzelnen Bewegungsmöglichkeiten hat sich hier der Begriff *Degree of Freedom* (DOF) etabliert.



(a) Skelett und *Joints* der Hand.

(b) DOF der *Hand Joints* (i.A.a. Erol u. a., 2007).

Abbildung 2.2: Skelett und DOF der Hand

Die einzelnen *DOF* der Hand können als **lokale Einflüsse** aufgefasst werden, die das Einnehmen einer bestimmten Handpose erlauben.
Neben den lokalen Einflüssen hängt die Darstellung der Hand noch von weiteren 6 *DOF* ab, die sich aus der räumlichen Translatation und Rotation der gesamten Hand zusammensetzen und als **globale Einflüsse** bezeichnet werden. Als Ursprung für die globalen *DOF* kann der Verbindungspunkt zwischen Handgelenk und Unterarm (*Wrist-Joint*) gesehen werden (s. Abb. 2.2).

Die Handkinematik ist mit den lokalen und globalen Einflüsse durch ca. 27 *DOF* gegeben. Die Menge aller DOF der Hand spannen dabei einen mehrdimensionalen stetigen Parameterraum auf. Eine konkrete Wertekonstellation dieses *DOF-Raums* beschreibt die relative Position jedes

Elementes der Hand und damit die Handpose eindeutig.
Das Ziel einer idealen Handposenerkennung kann daher so formuliert werden, dass der Wert jedes einzelnen *DOF* zu jeder Zeit aus einem gegebenen Eingabebild ermittelt werden kann. Das Erreichen dieses Ziels ist jedoch von einigen Schwierigkeiten begleitet, die im nächsten Unterabschnitt beschrieben werden.

2.1.1 Schwierigkeiten der visuellen Handposenerkennung

Neben den allgemeinen Schwierigkeiten, die bei der visuellen Erkennung von Objekten auftreten wie Störeinflüsse einer unkontrollierten Umgebung oder die hohe zu verarbeitende Datenmenge, identifizieren Erol u. a. (2007) und Yuan Yao und Yun Fu (2012) folgende drei Hauptschwierigkeiten der visuellen Handposenerkennung:

Selbstverdeckung (*Self-Occlusion*)
Wird die Hand lediglich aus einem definierten Blickwinkeln aufgenommen, kann die Hand als ganzes nicht dreidimensional[1] wahrgenommen werden. Dies führt dazu, dass Teile der Hand, die eine größere Entfernung zur Kamera aufweisen von näher liegenden Teilen verdeckt sein können und dadurch im Bild der Kamera nicht sichtbar sind (s. Abb. 2.3).

Abbildung 2.3: Handpose mit hohem Anteil an Selbstverdeckung

Mehrdeutigkeit (*Ambiguity*):
Die bildliche Darstellung unterschiedlicher Posen der Hand kann sich sehr stark ähneln. Dadurch ist es schwierig eine bildliche Darstellung der Hand eindeutig einer bestimmten DOF-Konstellation zuzuweisen. Dieser Umstand wird dabei durch die Selbstverdeckung noch begünstigt. Zudem können Störungen und Rauschen im Bild zu einer verfälschten Wahrnehmung der Hand führen und ebenfalls Mehrdeutigkeiten erzeugen.
Wird eine 2D-Kamera zur Aufnahme verwendet, wird das Problem der Mehrdeutigkeit durch die fehlende Tiefenwahrnehmung verstärkt (Shimada u. a., 1998), da hier während der Projektion, unterschiedliche Handposen auf annähernd die selbe Kontur abgebildet werden können (s. Abb. 2.4) und durch die fehlende Tiefeninformation nicht unterscheidbar sind (*Depth Ambiguity*).

[1] Aufnahme durch eine Kamera erlaubt maximal eine $2\frac{1}{2}$D-Wahrnehmung. Durch die Aufnahme über mehrere, um die Hand verteilte, Kameras kann die Wahrnehmung auf 3D erweitert werden.

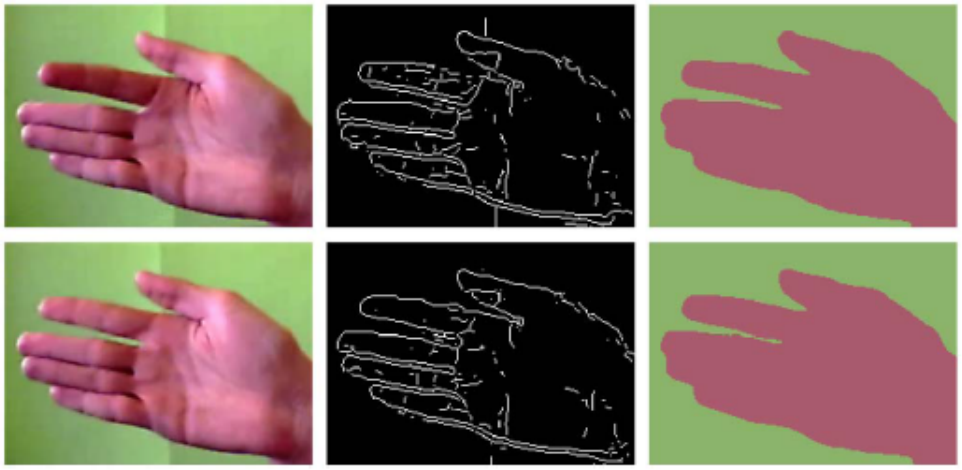


Abbildung 2.4: *Depth Ambiguity*: Mehrdeutigkeit der Handkontur bei veränderter Stellung des Zeigefingers (i.A.a. La Gorce u. a., 2011).

Hohe Dimensionalität:
Da die *DOF* sämtliche Bewegungsmöglichkeiten der Hand beschreiben, ist die Darstellung der Hand im Bild einer Kamera direkt von diesen abhängig. So können die *DOF* als Parameter einer idealen nichtlinearen Abbildungsfunktion betrachtet werden. Das Ziel der Handposenerkennung, das eine Bestimmung der DOF aus einer Darstellung bedeutet und damit eine Umkehrung der idealen Abbildungsfunktion darstellt, ist durch die fehlende Linearität und der zu bestimmenden DOF/Parameter-Anzahl von über 20 ein Problem sehr hoher Dimension, das nicht trivial gelöst werden kann.

2.2 Vorverarbeitung

Da die einzelnen Pixel eines Eingabebildes keine Zusammenhänge beschreiben, die als Information über den Wert des einzelnen Pixel hinaus zur Erkennung einer Handpose verwendet werden könnten, und weil die Betrachtung jedes einzelnen Pixel des Eingabebildes während der Posenerkennung meist mit einem höheren zeitlichen Aufwand verbunden ist, erfolgt in den meisten Lösungen eine Vorverarbeitung des Eingabebildes. Die wesentlichen Schritte der Vorverarbeitung sind dabei die Segmentierung potenzieller Pixelregionen[2] der Hand und die anschließende *Merkmalsextraktion* (s. Abb. 2.5).

[2]BLOB: Binary Large Opject

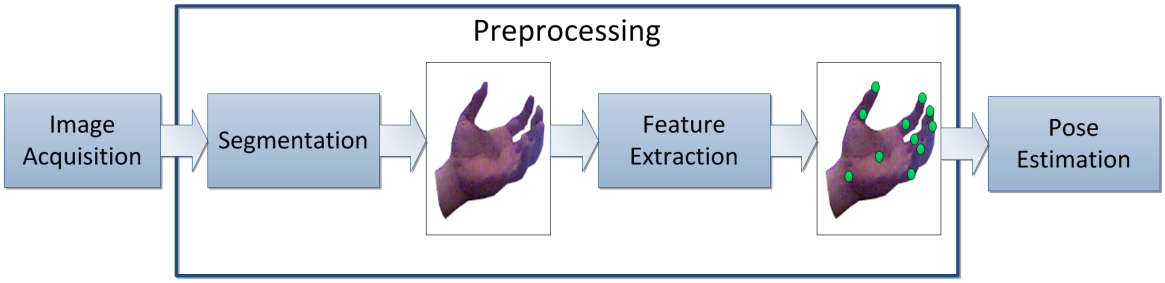


Abbildung 2.5: Schritte der Vorverarbeitung

2.2.1 Segmentierung der Hand

Verfahren zur Handposenerkennung setzen häufig voraus, dass potenzielle Handregionen aus dem Eingabebild bereits segmentiert wurden (vgl. Chan Wah Ng und Surendra Ranganath, 2002; Oikonomidis u. a., 2011b; Yuan Yao und Yun Fu, 2012), da dadurch einerseits die Menge der Eingabepixel auf eine überschaubare Anzahl reduziert wird und sich ein Zeitvorteil für weitere Verarbeitungsschritte ergibt. Andererseits erlaubt eine segmentierte Region, die Bestimmung von Konturmerkmalen.

Häufig wird zur Segmentierung die Hautfarbe als Merkmal genutzt (Zhigeng Pan u. a., 2010; Tofighi u. a., 2010; La Gorce u. a., 2011), da diese als pixelbasiertes Merkmal am ehesten die Charakteristik der Hand wiedergibt. Allerdings können dabei auch andere hautfarbene Bildregionen segmentiert werden (z.B. Gesicht), so dass in den weiteren Verarbeitungsstufen eine Unterscheidung zwischen den einzelnen Regionen erfolgen muss (s. Abb. 2.6). Eine Arbeit, die sich auf die Detektion und Verfolgung hautfarbener Regionen im Bild einer Kamera spezialisiert, liefern Argyros und Lourakis (2004). Eine Herausforderung bei der Verwendung der Hautfarbe als Merkmal besteht zudem darin, eine Invarianz gegenüber den unterschiedlichen menschlichen Hautfarben und äußeren Einflüssen der Umgebung zu erreichen (z.B. Beleuchtungsunterschiede). Dazu wird hier meist vorab ein generelles statistisches Hautfarbenmodell aus verschiedenen Bildern der menschlichen Haut trainiert, das dann in Kombination mit einem einfachen *Bayesian Classi-*

Abbildung 2.6: Segmentierung anhand der Hautfarbe (i.A.a. Zhigeng Pan u. a., 2010).

fier eine generelle Bestimmung hautfarbener Regionen ermöglicht ((Zhigeng Pan u. a., 2010)). Um die Genauigkeit der Hautfarbendetektion weiterhin zu erhöhen, wird in manchen Lösungen die Segmentierung um den Schritt einer, meist einfacher zu realisierenden, Gesichtserkennung erweitert (Alon u. a., 2009). Aus der gefundenen Region des Gesichtes lässt sich dann die Hautfarbe dynamisch bestimmen und kann zur Anpassung oder Ersetzung des Hautfarbenmodells verwendet werden.

Eine andere Möglichkeit zur Bestimmung der Vordergrundpixel ist die pixelbasierte Subtraktion des Eingabebildes mit einem vorab erstellten Hintergrundmodell (Brown und Capson, 2012; Ghosh u. a., 2010). Die verbleibenden Pixel des entstehenden Differenzbildes bilden dann die Vordergrundregionen (s. Abb. 2.7). Das Hintergrundmodell wird dabei vor der Erkennungsphase aus mehreren Aufnahmen des Hintergrunds gebildet (statisches Hintergrundmodell) was schließlich noch zu einem robusterne adaptives Hintergrundmodell erweitert (Ghosh u. a., 2010) erweitert werden kann.

Abbildung 2.7: Background Subtraction (QUELLE: http://e2e.ti.com). Links: Eingabebild. Rechts: Binarisiertes Differenzbild.

Wird ein Tiefensensor verwendet, kann zudem die Entfernung zum Sensor als Indikator zur Bestimmung der Handregion genutzt werden (Breuer, 2005; Gang Hu und Gao, 2011; Ren u. a., 2011)). Das Segment mit der geringsten Entfernung zum Sensor oder die Segmente innerhalb eines bestimmten Bereich werden dann als Kandidatsregionen gewählt (*Depth Thresholding*).

2.2.2 Extraktion von Merkmalen

Da die unkorrellierte Menge der Pixel eines Eingabebildes oder einer segmentierten Handregion nur wenig Informationen über die jeweilige Pose der Hand Preis geben, müssen hier einzelne Pixel oder Gruppen von Pixeln in Form von Merkmalen gefunden werden. Zwar beschreiben diese die Handpose nicht in eindeutiger Weise, erlauben in ihrer Gesamtheit aber eine bessere Einschätzung der Handpose. Zudem wird durch die Verwendung die Anzahl der zu verarbeitenden Daten erheblich reduziert.

Farbliche Regionale Merkmale

Eine einfache Möglichkeit, Zusammenhänge zwischen einzelnen Farbpixeln zu bilden, besteht darin, die Menge der Pixel in einzelne Region zu zerlegen und diese dann zu einem einzelnen Merkmal zusammenzufassen.

In diesem Zusammenhang stellen Viola und Jones (2001) einen Merkmalstyp vor, der in der Literatur häufig als *Haar-like features* bezeichnet wird und der sich als einfache pixel-summenbasiertes regionales Merkmal effektiv durch Integralbilder berechnen lässt.

Romero u. a. (2009) verwenden dagegen ein *Histogram of oriented gradients* (*HOG*) um die Pixel einer Kandidatsregion der Hand in einem Merkmal zusammenzufassen. Dabei wird ein Gradientenbild aus sämtlichen Regionspixel berechnet und aus diesem in verschiedenen Auflösungen ein Histogramm abgeleitet. Die Verkettung aller erstellten Histogramme ergibt dann einen Wert, der als *Descriptor* für die Region gelten kann.

Ein generelles Problem farblicher regionaler Merkmale ist, dass diese einerseits sehr schnell auf Veränderung der Beleuchtung reagieren, und dadurch mit einem hohen Rauschanteil gerechnet werden muss. Zum anderen verhindert die uniforme Farbgebung der Hand, dass von einer Farbregion auf die jeweilige Pose geschlossen werden kann. Nur die Kanten der Hand führen hier zu einen objektiv messbaren Unterschied. So wird meist auf geometrische formbeschreibende Merkmale zurückgegriffen (Stenger u. a., 2001; Chan Wah Ng und Surendra Ranganath, 2002; Ben Henia u. a., 2010).

Konturbasierte Merkmale

Die Kontur der Hand enthält bereits einige nützliche Informationen über die jeweilige Pose. Ist eine Handregion segmentiert, lässt sich aus dieser meist sehr einfach die Kontur bestimmen. Die Kontur kann dann meist durch einer geringe Anzahl von Deskriptoren z.B. durch *Fourier-Deskriptoren* (Chan Wah Ng und Surendra Ranganath, 2002; Wang u. a., 2007) beschrieben werden und so zur rudimentären Unterscheidung einzelner Posen bereits verwendet werden.

Um diese Unterscheidung anhand der Kontur unabhängig von der jeweiligen Lage und Größe der Hand durchführen zu können, kann eine invariante Beschreibung der Kontur durch die *Hu Invariant Moments* in Erwägung gezogen werden (Tofighi u. a., 2010).

Bei der ausschließlichen Verwendung der Kontur als Merkmal besteht allerdings die Schwierigkeit, mit der Mehrdeutigkeit umzugehen (s. Abb. 2.4 in Abschnitt 2.1.1). So eignen sich konturbasierte Merkmale hauptsächlich für partielle Erkennungsverfahren zur Unterscheidung einiger weniger Posen.

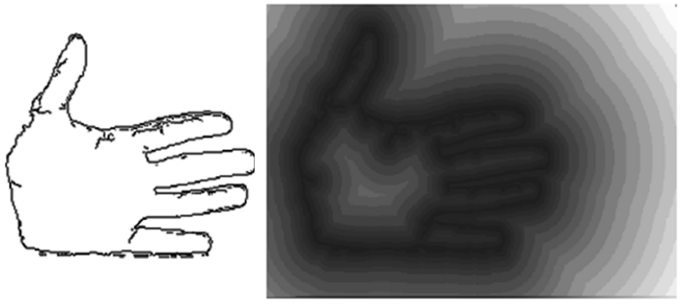
Kantenbasierte Merkmale

Um möglichst wenig von Mehrdeutigkeiten betroffen zu sein und einzelne Posen generell besser voneinander Unterscheiden zu können, wird bei einer zweidimensionalen Aufnahme meist auf kantenbasierte Merkmalstypen zurückgegriffen (Athitsos und Sclaroff, 2003; Ben Henia u. a., 2010; Oikonomidis u. a., 2011c). Die Verwendung von Kanten hat dabei den Vorteil, dass sich die einzelnen Kantenpixel sehr schnell durch einen gradientenbasiertem Filter aus einem Eingabebild extrahieren lassen.

Ein kantengefiltertes Eingabebild (Kantenbild) eignet sich dabei insbesondere zum effektiven Vergleich mit einem Referenzbild auf Basis einer Distanztransformation. In der Regel wird zum Vergleich die *Chamfer Distance Transformation* in mehr oder weniger angepasster Form verwendet (Athitsos und Sclaroff, 2003; Sudderth u. a., 2004; Stenger u. a., 2006), da diese eine einfache Berechnung der einzelnen Distanzen vorsieht.

Weiterhin bietet die Filterung von Kanten die Möglichkeit, innerhalb der Menge der Kantenpixel markante Punkte (*Keypoints*) mit einem höheren Informationsgehalt zu suchen. Eine sehr charakteristische Form von *Keypoints* stellen Ecken innerhalb der Kantenzüge dar. Kolsch und Turk (2004) und Zhigeng Pan u. a. (2010) nutzen bspw. die Merkmale des *Kanade-Lucas-Tomasi feature tracker* (*KLT-features* oder *good features to track*), um darauf aufbauend einzelne Eckpunkte zu bestimmen und darauf aufbauend die Bewegung der Hand im Bild zu verfolgen (*Tracking*). Da die natürliche Form der Hand jedoch eher wenig Ecken aufweist, eignen sich kurvenbasierte *Keypoints* eher um die Charakteristik der Hand wiederzugeben. So versuchen Gang Hu und Gao (2011) Wendepunkte und deren angrenzenden Kurvensegmente innerhalb der Kontur zu bestimmen, um diese als Merkmale für ihre partielle Posenerkennung zu verwenden.

Eine Schwierigkeit bei der Verwendung von kantenbasierten Merkmalen besteht jedoch darin, dass durch variierende Lichtverhältnisse und sonstige Störungen der Aufnahme Kanten extrahiert werden können, die keinem realem Kantenzug entsprechen (s. Abb. 2.8b). Dieser *Clutter* kann dabei zu Fehldeutungen der Handpose führen.



(a) Distanztransformation eines Kantenbildes (i.A.a. Ben Henia u. a., 2010).

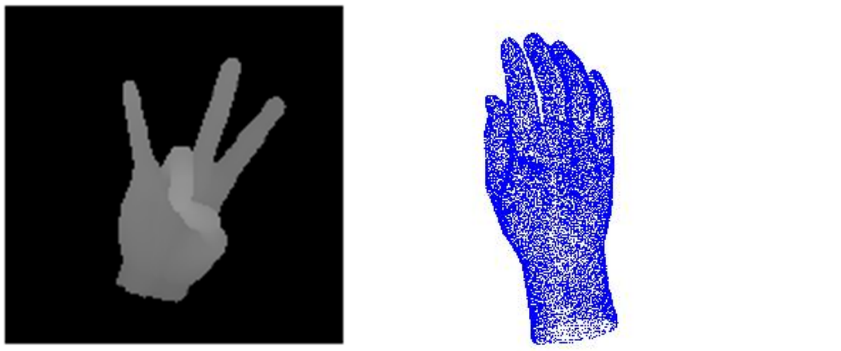
(b) Problem des *Kanten-Clutter* (i.A.a. Athitsos und Sclaroff, 2003).

Abbildung 2.8: Kanten als Merkmale

Tiefenmerkmale

Häufig finden sich Lösungen, die die farbliche 2D-Aufnahme durch eine Tiefenwahrnehmung ersetzen oder ergänzen (Breuer, 2005; Oikonomidis u. a., 2011b; Hamer u. a., 2009). Meist stehen die Tiefendaten ähnlich wie ein Farbbild in gerasterter Form zur Verfügung, so dass man hier von einem Tiefenbild sprechen kann (s. Abb. 2.9a).

Die Verwendung der Tiefe als Merkmal führt an sich bereits zu einer eindeutigeren Erkennung von Posen (s. *Depth Ambiguity* Abschnitt 2.1.1). Zudem liefern die einzelnen Pixel eines Tiefenbildes weit nützlichere Informationen als eine Menge von Farbpixeln. So ist die Merkmalsextraktion nicht nur auf einzelne Kanten beschränkt, sondern kann sich auf die gesamte Topologie der Handoberfläche beziehen. Ein Vergleich zweier Handregionen kann daher meist pixelweise erfolgen (Oikonomidis u. a., 2011b; Hamer u. a., 2009).
Ein weiterer Vorteil der Tiefenwahrnehmung ist ‚dass sich bei Kenntnis der Projektionsparameter die einzelnen Pixel des Tiefenbild in ihre realen räumlichen Koordinaten zurück transformieren lassen (Breuer, 2005). Die daraus entstehende 3D-Punktwolke kann dann zum räumlichen Vergleich mit einem 3D-Modell herhalten (s. Abb. 2.9b).



(a) Tiefenbild der Hand (i.A.a. Keskin u. a., 2011).

(b) Punktwolke. Quelle: http://www.farfieldtechnology.com

Abbildung 2.9: Interpretation von Tiefenwerten

2.3 Partial Hand Pose Estimation

Da eine generelle wertmäßige Bestimmung der *Hand-DOF* ein Problem sehr hoher Dimensionalität ist und daher meist zu einem hohen Aufwand während der Realisierung und innerhalb der Erkennungsphase führt, werden häufig Lösungen vorgeschlagen, die sich auf die Erkennung einzelner Teile der Hand und deren Bewegung spezialisieren, ohne die gesamte *DOF-Konfiguration* der Hand zu bestimmen. Andere Lösungen beschränken sich dagegen auf die

Erkennung einiger vorab definierter Posen (s. Abschnitt 2.3.2), wobei das Ziel ebenfalls weniger darin besteht, die gesamten *DOF* der Hand in jedem Aufnahmeschritt wertmäßig zu erfassen, sondern eher einzelne bekannte Posen als ganzes möglichst invariant im Eingabebild symbolisch wiederzuerkennen (Klassifizierungsproblem).
Diese eingeschränkte Erkennung kann dabei als partielle Posenerkennung (Erol u. a., 2007) aufgefasst werden und wird in der Literatur häufig auch als *Appearance-based* bezeichnet.

Das wesentliche Ziel der partiellen Posenerkennung ist, dass aus einem Eingabebild, einer bereits segmentierten Bildregionen oder der Menge extrahierter Merkmalen, neue Merkmale mit einem möglichst hohen Informationsgehalt (*High-level features*) extrahiert werden, die eine Bildregion eindeutig als Hand identifizieren. Dieser Schritt der Merkmalsextraktion bzw. Verfeinerung kann auch als Detektion bzw. Lokalisierung der Hand bezeichnet werden (s. Abschnitt 2.3.1). Da die Detektion der Hand wesentlich auf der Extraktion der starken Handmerkmale beruht, werden Verfahren der partiellen Posenbestimmung häufig auch als *Appearance-Based* bezeichnet. Schließlich können die extrahierten Merkmale, dann entweder direkt an eine Anwendung weiter gegeben werden oder zur Klassifizierung der Pose genutzt werden (s. Abschn, 2.3.2). Zudem kann die Bewegung der Hand anhand der Veränderung der Merkmalspositionen von Bild zu Bild durch ein *Tracking* (s. Abschn, 2.3.1) festgestellt werden, was sich insbesondere für die spätere Klassifizierung dynamischer Gesten eignet.

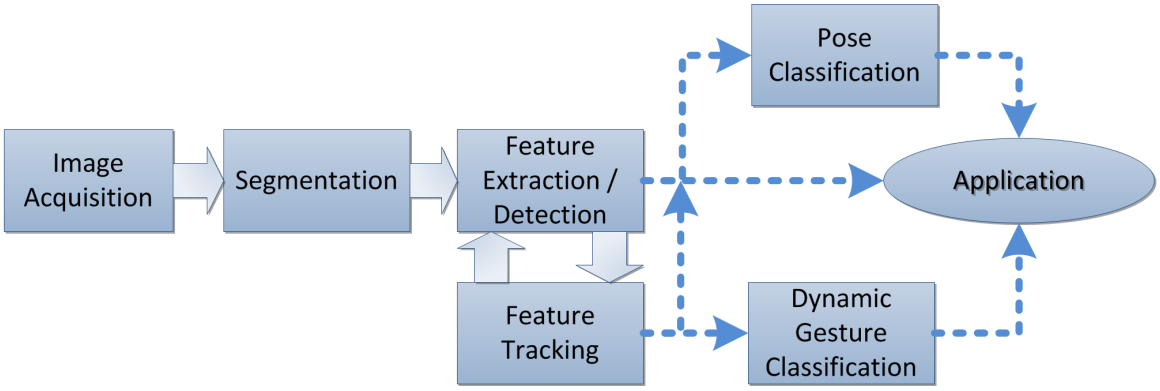


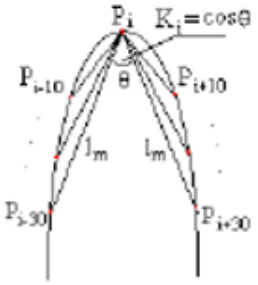
Abbildung 2.10: Schritte der partiellen Posenerkennung.

2.3.1 Detection und Feature-tracking

Zur genauen Detektion der Hand bedarf es eines möglichst eindeutiger Merkmale. So wird häufig versucht die Fingerspitzen oder den gesamten Finger als Merkmal zu bestimmen. Da die Extraktion dieser Merkmale häufig mit höheren Laufzeiten einhergeht, wird zudem häufig ein *Tracking*-Verfahren in Form eines dynamischen Filters eingesetzt, der aus der Merkmalsverteilung des aktuellen Bild Vorhersagen über die Verteilung im folgenden Bild trifft.

In der Literatur finden sich einige Arbeiten, die sich auf die ausschließliche Verfolgung der globalen Handbewegung konzentrieren.
So nutzen Zhigeng Pan u. a. (2010) einen speziellen Filter der anhand der Krümmung der Handkontur einzelne Kandidatenpixel, die auf Fingerspitzen hindeuten, selektiert (s. Abb. 2.11a). Zusätzlich wird das Zentrum der Handfläche über eine Distanztransformation der Handkontur detektiert[3]. Die So bestimmten Merkmale dienen dann zur Initialisierung eines KLT-Feature-Tracker (*Kanade- Lucas-Tomasi feature tracker*), der die Bewegung der Hand in den folgenden Aufnahmen verfolgt.

Ghosh u. a. (2010) nutzen eine Stereo-Kamera um die Bewegung der Hand im Raum zu verfolgen. Zur Detektion der Hand werden innerhalb der segmentierten Handregion die Lage einzelner Finger bestimmt. Dies erfolgt durch die Suche nach annähernd geraden Linien (s. Abb. 2.11b), die durch eine spezielle Approximation ermittelt werden.



(a) Bestimmung der Fingerspitzen auf Basis der Kontur-Krümmung (i.A.A Zhigeng Pan u. a., 2010).

(b) Detektion der Finger durch Linie-Approximation (i.A.A Ghosh u. a., 2010).

Abbildung 2.11: Detektion von Fingerspitzen als partielle Posenerkennung

Eine Lösung, die in der Lage ist auch komplexere lokale Bewegungen der einzelnen Finger zu erkennen und zu verfolgen stellen Hui Liang u. a. (2012) vor. Dazu werden aus der segmentierten Handregion eines Tiefenbildes die räumlichen Koordinaten in Form einer Punktwolke extrahiert. Basierend auf dem Konzept der *Accumulative Geodesic Extrema* werden nun die Punkte der einzelnen Fingerspitzen aus der Punktwolke ermittelt. Dabei werden jene Punkte als Fingerspitzen aufgefasst, die eine maximale *Geodätische Distanz* zum Zentrum der Handfläche aufweisen (s. Abb. 2.12). Da die Bestimmung der *Geodäti-*

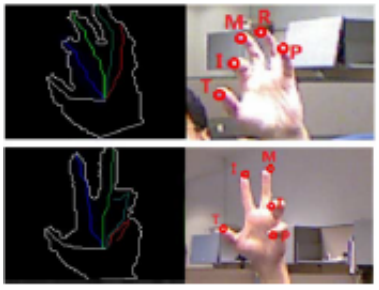


Abbildung 2.12: 3D-Detektion von Fingerspitzen durch *Accumulative Geodesic Extrema* (i.A.A Hui Liang u. a., 2012).

[3] Zentrum der Handfläche hat die höchste Entfernung zur Kontur.

sche Distanz für jeden Punkt sehr aufwändig[4] ist werden die gefundenen Fingerspitzen zur Initialisierung eines Tracking-Verfahrens in Form eines Partikelfilter genutzt, der in jedem weiteren Aufnahmeschritt geeignete Annahmen zur Positionen der Fingerspitzen im folgenden Bild trifft.

2.3.2 Erkennung vereinzelter Posen

Ein weiterer Schwerpunkt der partiellen Erkennung ist die Erkennung von vereinzelten, vorab definierten Posen. Die Art und Anzahl der zu erkennenden Posen hängt dabei von der jeweiligen Anwendung ab, was eine starke Spezialisierung der Posenerkennung auf das jeweilige Anwendungsgebiet zur Folge hat. Die Erkennung der Posen kann hier als Klassifizierungsproblem aufgefasst werden, da anhand der Merkmale eine Zuordnung zu einer entsprechenden Posenklasse erfolgt.

Chan Wah Ng und Surendra Ranganath (2002) errkennen vierzehn vorab unterscheidbarer Posen. Zur Erkennung der statischen Posen werden hautfarbene Bildregion in einem Segmentierungsschritt ermittelt und aus den Konturen der resultierenden Regionen die Fourier-Deskriptoren gebildet. Die Fourier-Deskriptoren werden dann durch einen Klassifizierer in Form eines *RBF - Network* (Radial Basis Funktion Network der jeweiligen Posenklasse zugeordnet.

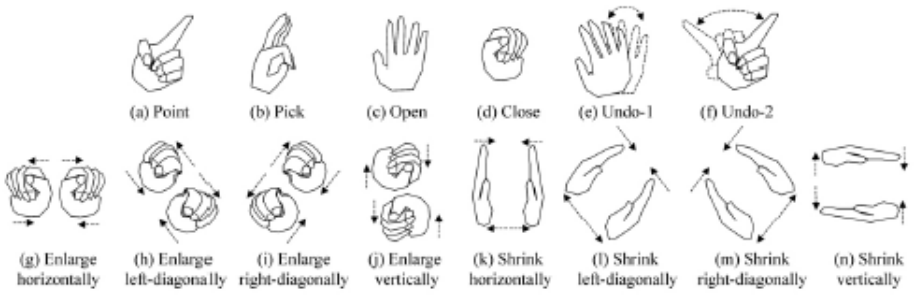


Abbildung 2.13: Vorab definierter Posensatz (i.A.A Chan Wah Ng und Surendra Ranganath, 2002).

Nguyen u. a. (2011) bestimmen dagegen aus dem Eingabebild *Haar-like Feature* (s. Abschnitt 2.2.2). Die extrahierten Merkmale werden dann durch mehrere *Cascaded Adaboost Classifier* einer von zwölf Posenklassen zugeordnet.

Tofighi u. a. (2010) nutzen als Merkmal sog. *Convexity Defects* (s. Abb. 2.14), um grob zwischen möglichen und nicht möglichen Klassen von Posen zu unterscheiden. Zur genaueren Zuordnung zu einer Posenklasse werden dann aus der Kontur der Hand die invarianten Hu-Momente gebildet und mit den Hu-Momenten der zu erkennenden Posen verglichen.

[4] Graph bilden und Ausführung des Dijkstra-Algorithmus

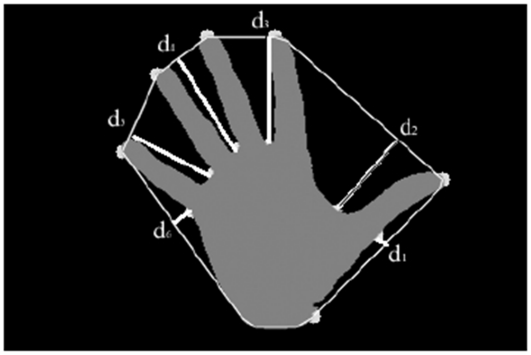


Abbildung 2.14: *Convexity Defects* (i.A.a. Tofighi u. a., 2010). Große Abstände zwischen der Kontur und deren Konvexehülle deuten auf Fingerpositionen hin.

2.3.3 Partial Hand Pose Estimation Fazit

Allgemein lässt sich für partielle Verfahren festhalten, dass die Beschränkung der Erkennung auf relevante Teile der Hand, globale Bewegungen oder einzelne Posen dazu führt, dass die Problemdimensionalität stark reduziert wird. So können partielle Verfahren meist mit geringerem Aufwand und oft sehr performant realisiert werden. Die Erkennung in Echtzeit mit Detektionsraten von über 60 Hz ist daher keine Seltenheit.

Ein wesentlicher Nachteil besteht jedoch in der mangelnden Generalität der Verfahren. So führt die Spezialisierung auf einzelne Teile oder Posen der Hand dazu, dass nicht der gesamte Zustand der Hand betrachtet wird. Ein Sachverhalt der den Umgang mit Selbstverdeckungsfällen und Mehrdeutigkeiten erschwert und das Risiko für Fehldetektionen erhöht. So werden vor der Realisierung eines partiellen Verfahren häufig Randbedingungen im Umgang mit der Anwendung aufgestellt, die den Benutzer zwingen nur bestimmte Handbewegungen durchzuführen (Erol u. a., 2007), wodurch jedoch nur eine eingeschränkte Interaktion mit der jeweiligen Anwendung erlaubt werden kann.

2.4 Single View Pose Estimation

Um im Gegensatz zu partiellen Verfahren die kompletten *DOF* der Hand im Sinne einer *Full DOF Hand Pose Estimation* möglichst generell und kontinuierlich in jedem Aufnahmeschritt bestimmen zu können müssen andere Lösungsansätze verfolgt werden. Diese lassen sich laut Erol u. a. (2007) in die ***Single View Pose Estimation*** **und das** ***Model-based Tracking*** unterteilen.

Single View Pose Estimation-Ansätze zeichnen sich dadurch aus, das aus einem einzelnen Eingabebild bzw. aus den extrahierten Merkmalen die kompletten *DOF* der Hand direkt wertmäßig erfasst werden können. Dazu bedarf es einer Zuordnung der, aus dem einzelnen Aufnahmebild extrahierten, Merkmale zu einer Wertekonstellation im *DOF-Raum*. Diese Anforderung ähnelt dabei stark der für die Erkennung einzelner Posen im Bereich der *Partial Hand Pose Estimation* (s. Abschnitt 2.3). Allerdings muss hierbei, zur Wahrung der Generalität, möglichst jede Pose der Hand wiedererkannt werden.
Eine Zuordnung zwischen Merkmalen und Werten der *DOF* wird meistens durch ein diskretisierendes exempelbasiertes Verfahren hergestellt. Die genutzten Verfahren lassen sich dabei grob in die Kategorien des *Template Matching* (s. Abschn. 2.4.1), die *Klassifizierung* (s. Abschn. 2.4.2), und die *Regression* (s. Abschn. 2.4.3) unterteilen.

2.4.1 Single View Pose Estimation durch Template Matching

Eine Möglichkeit der Zuordnung zwischen Merkmalen und der jeweiligen *DOF-Konstellation* kann durch den direkten Vergleich der Merkmale mit in der Anwendung hinterlegten Referenzbildern bzw. -Merkmalsverteilungen (*Templates*) erfolgen, für die die jeweilige DOF-Konstellation bekannt ist. (Athitsos und Sclaroff, 2003; Romero u. a., 2009). Die gesamte Menge aller *Templates* mit ihren zugehörigen *DOF-Konstellation* wird dabei in geeigneter Weise (häufig in Form einer Datenbank) dem Prozess der Posenerkennung zur Verfügung gestellt.
Während der Erkennungsphase wird dann das *Template* mit der höchsten Übereinstimmung gegenüber den extrahierten Merkmalen des Eingabebildes über die Menge aller *Templates* gesucht und deren *DOF-Konstellation* als Ergebnis der Posenerkennung ausgegeben.
Um eine sehr hohe Erkennungsgenauigkeit zu erreichen, ist die erforderliche Anzahl der Templates meist sehr hoch[5]. Da eine Suche nach einem korrespondierenden *Template* über die gesamte Menge durch den direkten Vergleich zeitlich nicht unerheblich ist und zudem alle *Templates* häufig nicht komplett im Hauptspeicher hinterlegt werden können, besteht eine Hauptanforderung darin den Vergleich mit allen *Templates* möglichst effektiv zu gestalten und/oder Indizierungsverfahren zu entwickeln die auf Basis der extrahierten Merkmale ein korrespondierendes *Template* direkt bestimmen.

Athitsos und Sclaroff (2003) beschreiben ein *Template Matching* auf Basis eines kantengefiltertem Eingabebildes. Dazu wird vor der Erkennung eine große Menge an Referenzbildern synthetisch erzeugt. Aus diesen werden dann einzelne markante Linie extrahiert und im Hauptspeicher hinterlegt. Neben diesen Linien wird aus den Bildern eine Approximierte *Chamfer Distanz Transformation* gebildet, deren Ziel eine Reduktion der Dimension der Distanztransformierten durch die Einbettung der einzelnen Distanzen in einen K-Dimensionalen-Euklidischen Raum ist. Durch diese Dimensionsreduzierung lässt sich die Distanztransformierte approximiert durch einen K-Dimensionalen-Vektor ausdrücken, und kann so ebenfalls, den einzelnen Linien-Templates zugeordnet, im Hauptspeicher hinterlegt werden. Durch den Vergleich des Eingabe-

[5] 107328 bei Athitsos und Sclaroff (2003)

bildes mit allen approximierten Distanztransformierten können nun einzelne Posen-Kandidaten bestimmt werden. Zur Selektion des besten Kandidaten wird dann das Eingabekantenbild mit den entsprechenden Linien der Kandidaten verglichen (s. Abb. 2.15).
Athitsos und Sclaroff (2003) betonen jedoch dass das vorgestellte Verfahren eine zu geringe Erkennungsgenauigkeit aufweist um als eigenständige Posenerkennung zu fungieren, sich allerdings zur einfachen Initialisierung eines z.B. modellbasierten Hand-Tracking-Verfahren verwenden lässt.

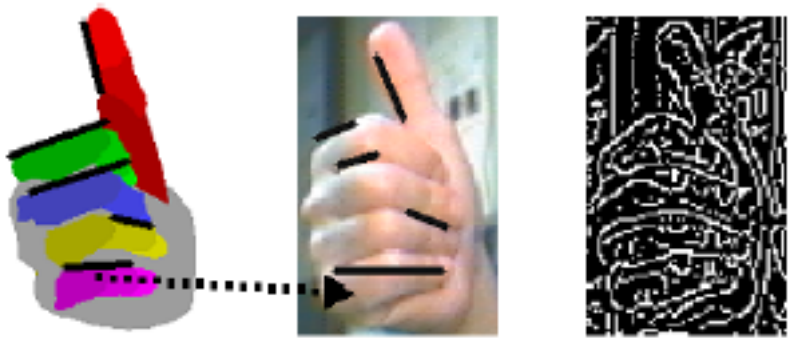


Abbildung 2.15: Illustration des *Line Matching* Verfahrens von Athitsos und Sclaroff (2003), das zur genaueren Selektion eines Templates verwendet wird. Dabei werden gemeinsame gerade Linien aus einem Referenzbild (links) und einem Kantenbild der Aufnahme (rechts) ermittelt. Das mittlere Bild zeigt exemplarisch die Ergebnisse des *Line Matching*.

Ein generelles Problem von *Single View Pose Estimation*, das sich insbesondere beim Template-Matching bemerkbar macht, ist, dass durch die statische Zuordnung zwischen extrahierten Merkmalen zu einzelnen DOF-Konstellation die Erkennung meist stark von der Mehrdeutigkeit betroffen ist. So verweisen extrahierte Merkmale nicht immer eindeutig auf eine einzelne DOF-Konstellation.
Um eine eindeutigere Zuordnung zwischen extrahierten Merkmalen und DOF zu erreichen erweitern Romero u. a. (2009) ihr datenbankbasiertes *Template Matching* um ein einfaches Tracking. Aus einer segmentierten Handregion wird dazu das *Histogram of Orientated Gradiants* (*HOG*) (s. Abschnitt 2.2.2) aus einer Handregion erstellt, welches zur Indizierung eines Templates in der Datenbank genutzt wird. Die Indizierung erfolgt dabei über ein Verfahren namens *Locality Sensitive Hashing* (*LSH*) , das die Indizierung (*Nearest-Neighbour Search*) einer ‚über die Einträge der Datenbank erlernten Suchfunktion mit geringerem Aufwand ermöglicht (sublineare Komplexität). Da eine Handpose aufgrund der Mehrdeutigkeit jedoch nicht allein durch den HOG-Merkmalstyp unterschieden werden kann und daher das Ergebnis der Indizierung aus mehreren Templates besteht, wird hier die zeitliche Kontinuität zur endgültigen Bestimmung der DOF aus den indizierten Templates mit einbezogen (s. Abb. 2.16).
Die Einbeziehung der Dynamik führt dazu, dass das beschriebene Verfahren weniger als *Single View Pose Estimation*-Verfahren angesehen werden kann sondern eher als Tracking gelten kann.

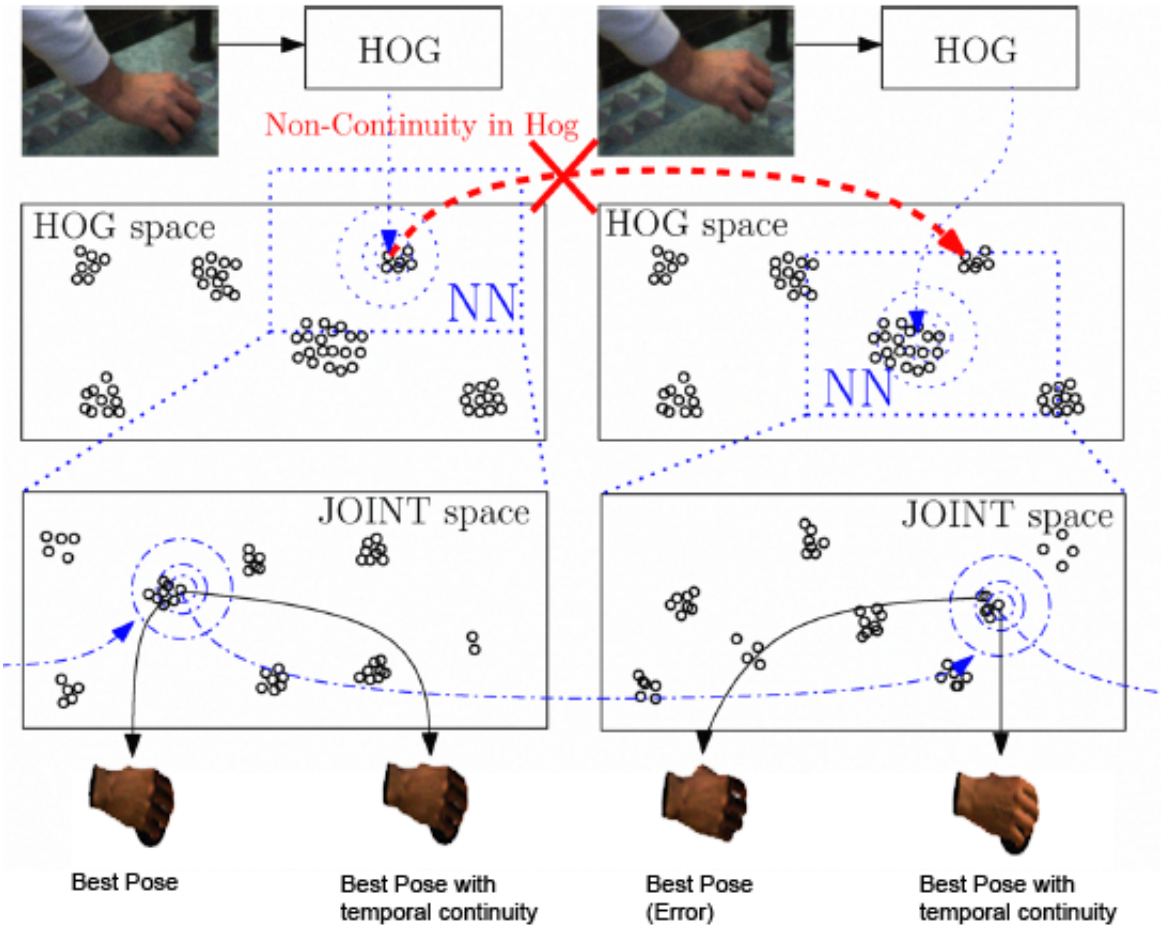


Abbildung 2.16: Skizze des Problems der Mehrdeutigkeit und deren Lösung durch Einbeziehen der Dynamik in der Template-basierten Lösung von Romero u. a. (2009). Die Suche nach einem korrespondierenden Template durch den Vergleich von HOG-Merkmalen eines Eingabebildes (oberste Zeile) ergibt aufgrund der Mehrdeutigkeit mehrere Möglichkeiten (zweite Zeile). Daher werden auch mehrere Konstellationen im DOF-Raum (dritte Zeile, hier als Joint Space bez.) gefunden. Die Auswahl eines konkreten Ergebnisses aus dieser Menge der Konstellationen kann dabei ohne Berücksichtigung der Handdynamik zu fehlerhaften Erkennungsresultaten führen (Ergebnisbild unterste Zeile, dritte Spalte). Eine Einbeziehung der Dynamik kann dabei nur im DOF-Raum erfolgen, da im HOG-Merkmalsraum keine temporalen Zusammenhänge bestehen (s. roter Pfeil).

2.4.2 Single View Pose Estimation durch Klassifizierung

Eine *Single View Pose Estimation* kann zudem durch eine Klassifizierung erfolgen. Da die Hand eine sehr hohe Anzahl von möglichen Posen aufweist und zudem Mehrdeutigkeiten auftreten, ist es hier häufig nicht ratsam einen Klassifizierer, für die gesamte Konfiguration der Hand zu verwenden. So müsste das jeweilige Klassifizierungsverfahren mit einer sehr hohen Anzahl von Posenklassen umgehen und zudem die einzelnen Mehrdeutigkeiten behandeln, was letztendlich zu einer Erhöhung des Trainings- und Klassifizierungsaufwand führt. Die Klassifizierung besteht daher eher in der Einordnung der Merkmale des Eingabebildes in Klassen, die jeweils ein Teil der Hand repräsentieren. Die Identifikation der einzelnen Teile im Bild erlaubt dann eine leichtere Bestimmung der *DOF* in einem zusätzlichen Schritt.

Inspiriert durch die Lösung zur Körperposenerkennung von Shotton u. a. (2011) nutzen Keskin u. a. (2011) *Random Trees* bzw. einen *Random Forest*[6] zur Handposenerkennung aus dem Tiefenbild eines Kinect-Sensors. Der *Random Forest* ,der sich hier aus mehreren Klassifizierungsbäumen zusammensetzt, wird dabei genutzt, um die einzelnen Pixel einer aus dem Tiefenbild segmentierten Handregion Klassen zuzuordnen, die die einzelnen Teile der Hand repräsentieren (s. Abb. 2.17). Sind die einzelnen Pixel der Handregion klassifiziert, kann die Menge der Pixel einer Klasse als Segment des jeweiligen Handteils betrachtet werden. Um die *DOF* der Hand zu beschreiben, wird aus den erkannten Segmenten ein Skelett gebildet. Dazu werden die Mittelpunkte der Segmente bestimmt, die als einzelne Verbindungspunkte (Joints) des Handskelletes gelten, und entsprechend miteinander verbunden.
Diese vorgeschlagene Methode ermöglicht dabei sehr schnelle Erkennungsraten, allerdings sind die die reinen Pixelwerte ein Merkmal mit sehr geringen Informationsgehalt, so dass dies durch eine hohe Anzahl an Trainingsdaten für den Klassifizierer kompensiert werden muss.

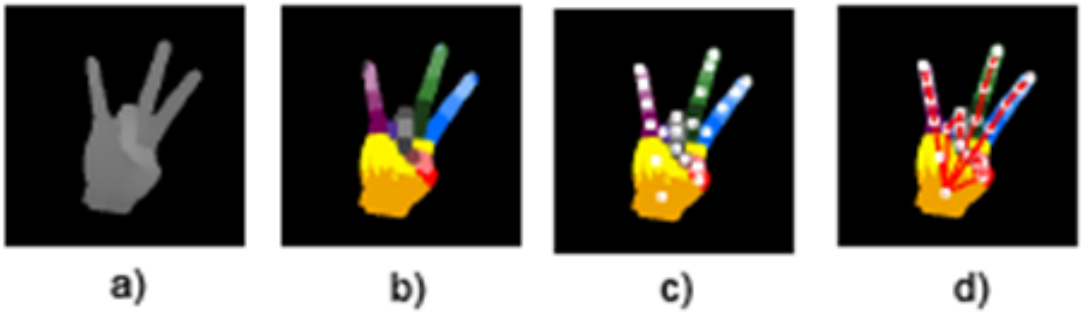


Abbildung 2.17: Klassifizierung einzelner Handteile zur Bestimmung der DOF nach Keskin u. a. (2011). a) Segmentiertes Tiefenbild. b) Eingeordnete Teile der Hand (*Labeling*) c) Extrahierte der Verbindungspunkte (Joints) der Teilsegmente. d) Verbindung der einzelnen Verbungspunkte zu einem Skelett.

[6] *Random Forest*: Klassifizierer, der sich aus mehreren Klassifizierungsbäumen zusammensetzt, die eine Einordnung in äquivalente Klassen vornehmen. Die Zusammenfassung der Klassifizierungsergebnisse der einzelnen Bäume lässt dann eine Aussage über die jeweilige Klasse zu. Die einzelnen Bäume werden durch einen, über Zufallswerte angepassten, Trainingsdatensatz erstellt und berücksichtigen daher die natürliche Varianz der Merkmalsverteilungen.

Ein ähnliches Verfahren wie Keskin u. a. (2011) beschreiben Yuan Yao und Yun Fu (2012) in ihrer Arbeit. Dabei nutzen sie ebenfalls einen *Random Forest* um eine Zuordnung von Tiefenpixeln einer Handregion zu den jeweiligen Teilen der Hand zu erreichen. Allerdings werden in dem beschriebenen Verfahren Gradienten der Handoberfläche und die räumliche Position der Pixel in Form einer Punktwolke als Merkmal zur Klassifizierung der einzelne Teile verwendet, was zu einer robusteren Klassifizierung mit geringerem Trainingsaufwand führen sollte.

Ein Grundsätzliches Problem bei der Klassifizierung einzelner Teil der Hand besteht darin, ähnlich wie bei partiellen Verfahren, stark von Selbstverdeckungsfällen betroffen zu sein. Da sich bspw. die Form der einzelnen Fingerglieder sehr stark ähnelt muss der Klassifizierer die relative Position zu andern Teile zur Bestimmung eines einzelnen Fingergliedes mit einbeziehen. Sind mehrere Teile der Hand durch Selbstverdeckung im Bild nicht sichtbar besteht daher die Möglichkeit das einige Bildteile der Hand nicht richtig eingeordnet werden. So bedarf es in diesen Lösungen meist einer expliziten Behandlung der Selbstverdeckung (Yuan Yao und Yun Fu, 2012).

2.4.3 Single View Pose Estimation durch Regression

Die Regression der Abbildungsfunktion zwischen dem Merkmalsraum und dem DOF-Raum, stellt eine weitere Möglichkeit für eine *Single View Pose Estimation* dar. Im Gegensatz zu den Template-basierten Verfahren, die nur eine diskrete Zuordnung zwischen Eingabemerkmalen und DOF-Konstellation durch die endliche Anzahl an Templates erlauben, erlaubt die Regression der Abbildungsfunktion prinzipiell auch eine stetige Zuordnung.

Zur Regression der Abbildungsfunktion verwenden Rosales u. a. (2001) ein nichtlineares trainingbasiertes Verfahren namens SMA *Specialized Mapping Architecture*. In der vorgeschlagenen Lösung erfolgt ein überwachtes trainieren der SMA durch einen Datenhandschuh, über den einzelne DOF-Konstellationen der Hand ermittelt und zur Anpassung eines grafischen 3D-Modells der Hand verwendet werden. Aus dem Bild lassen sich dann Merkmale bestimmen und direkt den, über den Handschuh gemessenen, DOF-Werten zuordnen. Aus den so gewonnenen Trainingsdaten wird dann eine Menge statistischer Funktionen[7] hergeleitet, die zusammen eine Zuordnung der Merkmale zu einer passenden DOF-Konstellation ermöglichen. Jede Funktion spezialisiert sich dabei auf die Einteilung eines Bereichs im Merkmalsraum, den sie besser zuordnen kann als die anderen. Während der Erkennung werden die extrahierten Merkmale des Aufnahmebildes an die Funktionen übergeben, die jeweils einen Vorschlag für die zugehörige DOF-Konstellation hervorbringen. Die einzelnen Vorschlägen der Funktionen werden gemäß der jeweiligen Funktionsspezialisierung gewichtet. Der Vorschlag mit der höchsten Gewichtung dient schließlich als Endergebnis der Posenerkennung.
Die Erprobung der vorgestellte Methode erfolgt allerdings ausschließlich für synthetischen Da-

[7] Es müssen mehrere Funktionen sein, da eine einzelne Funktion keine Auflösung der Mehrdeutigkeiten erlaubt.

ten, so dass wenig Informationen über den praktischen Einsatz insbesondere in Bezug auf singuläres Verhalten zur Verfügung stehen.

2.4.4 Single View Pose Estimation Fazit

Ein generelles Problem dem die *Single View Pose Estimation* gegenüber steht ist die Mehrdeutigkeit. So kann die Abbildung zwischen Merkmalen und einzelnen DOF als stark nichtlineare N:N-Beziehung aufgefasst werden. Erfolgt keine Behandlung der Mehrdeutigkeit kann dies bei einer kontinuierlichen Erkennung schnell zu Fehlern führen. Zur Behandlung der Mehrdeutigkeit bliebt häufig keine andere Wahl, als die zeitliche Kontinuität durch ein Tracking in die Posenbestimmung mit einzubeziehen (Erol u. a., 2007; Stenger, 2004; Romero u. a., 2009). Dennoch sind Verfahren im Bereich der *Single View Pose Estimation* meist zur schnellen und einfachen Initialisierung eines Tracking-Verfahrens unabdingbar.

Weiterhin lässt sich für exempelbasierte Verfahren der *Single View Pose Estimation* festhalten, dass sich der, für eine Posenerkennung nötige, Berechnungsaufwand (Computional Complexity) während der Laufzeit durch die *offline* Vorbereitung, in Form einer Trainingsphase oder der Erstellung einer *Template-Datenbank*, massiv reduzieren lässt. Dem reduzierten Berechnungsaufwand steht jedoch der hohe Erstellungsaufwand gegenüber. Um eine robuste nahezu generelle Posenerkennung zu erreichen, muss eine hohe Anzahl an Exempel bzw. *Samples* erstellt werden, was meist nur mit Hilfe eines grafischen Handmodells und eines automatisierten Syntheseprozesses möglich ist (Athitsos und Sclaroff, 2003; Romero u. a., 2009; Yuan Yao und Yun Fu, 2012). Dadurch werden jedoch die Variationen, denen eine realistische Abbildung von Merkmalen auf einzelne *DOF* unterliegt, ignoriert.

Unabhängig von dem hohen Erstellungsaufwand ist ein wesentlicher Nachteil von *Single View Pose Estimation* dadurch gegeben, dass eine exakte Zuordnung der Eingabemerkmale zu den jeweiligen *DOF* nur für die tatsächlich vorhanden *Posen-Sample* bekannt ist. Da jedoch die Anzahl der *Sample* endlich ist, kann der DOF-Raum exakt nur in diskreter Weise beschrieben werden. Zwischen den einzelnen *Samplen* kann ein Verfahren daher im besten Fall nur approximierte Antworten als Resultat der Posenerkennung liefern (s. Abb. 2.18).
Dieser Umstand führt dazu, dass die Genauigkeit und Generalität der jeweiligen Single View Pose Estimation direkt von der Anzahl der *Samples* und deren jeweiliger Verteilung im DOF-Raum abhängt, wobei allerdings eine stetige exakte Bestimmung sämtlicher Handposen nie garantiert werden kann (Erol u. a., 2007).

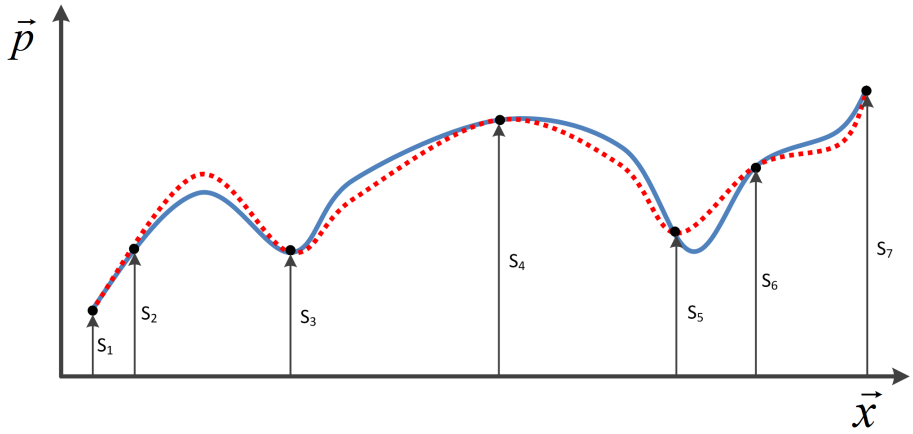


Abbildung 2.18: Darstellung der Abhängigkeit von einzelnen Samples bei der Single View Pose Estimation. Die ideale Abbildungsfunktion (blaue Linie) zwischen einer jeweiligen Merkmalsverteilung $\vec{x}$ und einer DOF-Konstellation $\vec{p}$ kann nur an den Stützstellen der gegebenen Samples S_i exakt beschrieben werden. Zwischen diesen Samplen muss die ideale Abbildung approximiert werden (rote Linie). Die Qualität der Approximation hängt neben der Anzahl der Samples insbesondere von deren Verteilung ab.

2.5 Model-based Pose Tracking

Der zweite Ansatz der *Full DOF Hand Pose Estimation* kann als *Model-based tracking* bezeichnet werden (Erol u. a., 2007).

Im Gegensatz zu den exempelbasierten Methoden der *Single View Pose Estimation*, wird bei dem modellbasiertem Ansatz ein parametrisiertes Modell der Hand genutzt, um dieses zur Laufzeit an die bildliche Darstellung der Hand im Eingabebild anzupassen. Dazu gilt es zunächst die Form des Modells mit Hilfe eines geeigneten **Strukturmodells** festzulegen (s. Abschnitt 2.5.1). Das Strukturmodell sollte dabei die reale Form der Hand möglichst exakt nachbilden, um während der Posenerkennung die jeweilige Darstellung des Modells möglichst genau mit der bildlich wahrgenommen Hand vergleichen zu können. Die im Strukturmodell beschriebene Form der Hand lässt sich dabei durch die Veränderung einzelner Modellparameter (bzw. *Modell-DOF*) dynamisch anpassen.

Die Art, wie die Transformation des Strukturmodells über die einzelnen Parameter vollzogen wird, wird durch ein **kinematisches Modell** vorgegeben (s. Abschnitt 2.5.2). Meistens wird hier ein kinematisches Modell eingesetzt, das die Bewegung des realen Skelettes einer Hand nachempfindet, so dass die Modellparameter die einzelnen DOF der Hand widerspiegeln (La Gorce u. a., 2011; Stenger u. a., 2001; Oikonomidis u. a., 2011b). Dies ist allerdings nicht zwingend

erforderlich, wie später noch gezeigt wird (s. Abschn. 2.5.2).

Durch die Anpassbarkeit des formbeschreibenden Strukturmodells über das kinematische Modell ist es mit Hilfe von mehr oder weniger angepassten Darstellungsverfahren (Rendering) möglich, zur Laufzeit Modellmerkmale zu generieren[8]. Die generierten Merkmale können dann zum direkten Vergleich mit den extrahierten Merkmalen aus dem Eingabebild herangezogen werden. Die Kombination von Transformation und Merkmalsgenerierung kann so formal als Funktion M betrachtet werden, die in Abhängigkeit von den Modellparametern $\vec{p_m}$ den vergleichbaren Modellmerkmalsvektor $\vec{x_m}$ erzeugt.

$$\vec{x_m} = M(\vec{p_m}) \quad \text{mit} \quad \vec{x_m} = \{x_{m1}, x_{m2}, \ldots, x_{mn}\} \tag{2.1}$$

Durch die Vergleichbarkeit von extrahierten Bild- und generierten Modellmerkmalen, ist es nun möglich eine Fehlerfunktion oder Distanzfunktion E aufzustellen, die den Unterschied (Fehler) e zwischen dem Eingabemerkmalsvektor $\vec{x}$ und den, in Abhängigkeit von $\vec{p_m}$ erstellten, Modellmerkmalsvektor $\vec{x_m}$ misst.

$$e = E(\vec{x}, \vec{x_m}) = E(\vec{x}, M(\vec{p_m})) \tag{2.2}$$

Die Art und Weise, wie eine Fehlerfunktion realisiert wird, ist stark von dem jeweiligen Verfahren und den darin verwendeten Merkmalen abhängig. Die Definition dieser Funktion und die Wahl der Bild- und Modellmerkmale wird dabei in der Literatur häufig unter dem Begriff des **Observationsmodells** zusammengefasst (s. Abschnitt 2.5.3).

Ist der Unterschied zwischen einer bestimmten Modellkonfiguration und einem gegebenen Eingabebild durch ein geeignetes Observationsmodell messbar, besteht das Ziel der modellbasierten Posenerkennung darin in jedem Aufnahmeschritt eine Modellparameterkonstellation zu finden, die den kleinsten Übereinstimmungsfehler zwischen Modell- und den extrahierten Merkmalen des Aufnahmebildes liefert. Die einzelnen Modellparameter beschreiben dann die jeweilige Konfiguration einer wahrgenommenen Hand und können daher als Resultat der Posenerkennung gelten.
So lässt sich die Posenerkennung als Suche nach einer Wertekonstellation im stetigen hochdimensionalen Modellparameterraum auffassen, die einen minimalen Fehler gegenüber einer observierten Hand verursacht. Dies stellt eine harte Optimierungsaufgabe dar. Zudem besteht auch hierbei das Problem, dass durch die Mehrdeutigkeit mehrere bzw. falsche Modellkonfiguration einen minimalen Fehler gegenüber den extrahierten Merkmalen liefern können (Erol u. a., 2007). Zur Auflösung der Mehrdeutigkeit wird daher meist die zeitliche Kontinuität genutzt und durch ein **Tracking** (s. Abschnitt 2.5.4) in Form eines dynamischen Filters in die Erkennung mit einbezogen. Die Verwendung eines dynamischen Filters hat auch den Vorteil, dass auf Basis vergangener Ergebnisse, Vorhersagen für die Parameterkonstellation in der aktuellen Aufnahme getroffen werden können, was die Suche allgemein vereinfacht.

Bei Verwendung eines *Tracking* besteht eine zusätzliche Anforderung darin, mit Situationen, in

[8] Diese Synthese der Merkmale führt dazu, dass modellbasierte Ansätze in der Literatur häufig auch als Generativeverfahren bezeichnet werden.

denen keine Historie von Erkennungsergebnissen vorhanden ist, umzugehen. Daher muss ein zusätzlicher Schritt in Form einer **Initialisierung** des *Tracking* (s. Abschnitt 2.5.5) im Prozess einer modellbasierten Posenerkennung vorgesehen werden (s. Abb. 2.19).

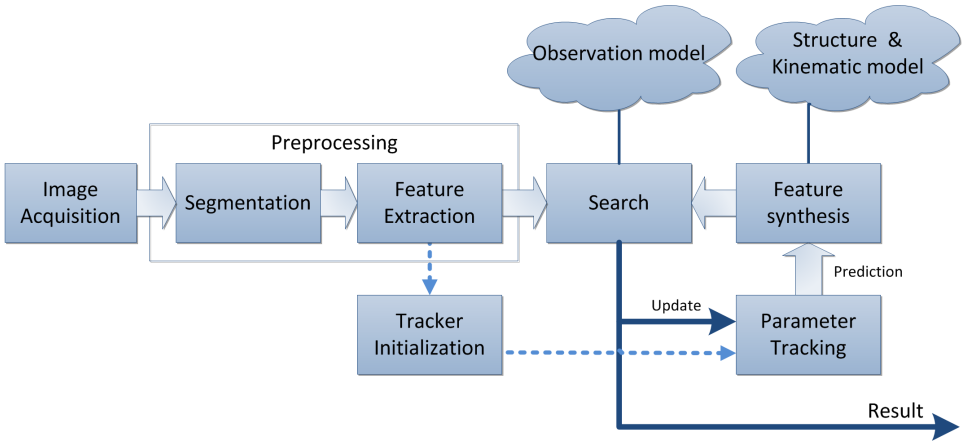


Abbildung 2.19: Essenzielle Schritte einer modellbasierten Posenerkennung und deren genutzter Modelle. Anzumerken sei, dass das Tracking eine Initialisierung im ersten Bild der Aufnahme und bei Unterbrechung der Verfolgung benötigt, da hier keine vorherigen Resultate ausgewertet werden können.

2.5.1 Strukturmodell

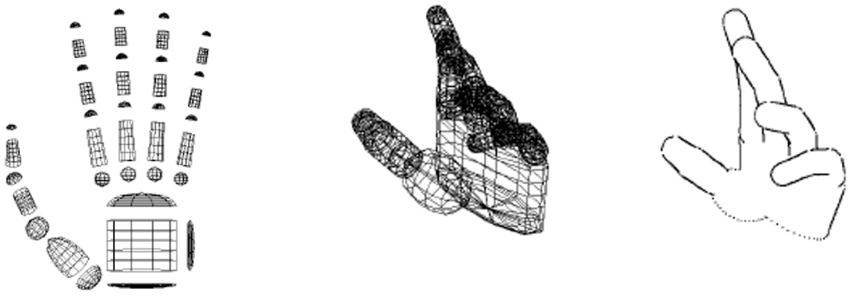
Da das kontinuierliche Transformieren, Generieren von Merkmalen und Vergleichen den Hauptaufwand der modellbasierten Posenerkennung ausmacht und damit die Erkennungsgeschwindigkeit bestimmt, orientiert sich die Wahl des Strukturmodells häufig daran, wie einfach dieses sich unter diesen Gesichtspunkten verarbeiten lässt.
Um die komplette räumliche Handpose zu beschreiben, wird meist ein grafisches 3D-Modell der Hand verwendet[9] , wobei dieses sich entweder aus mehreren unabhängigen 3D-Primitiven zusammensetzt oder durch ein einheitliches *Polygon Mesh* modelliert werden kann. Für den bildlichen Vergleich zwischen der Darstellung der Hand und der des Modells erfordert die Merkmalsgenerierung jedoch eine Projektion des Strukturmodells in die Ebene des Eingabebildes (Stenger u. a., 2001; Oikonomidis u. a., 2011c). Die Projektion muss dabei die reale Projektion der verwendeten Kamera möglichst exakt nachempfinden, um das Modell auf die gleichen Pixel abzubilden wie die der realen Hand im Eingabebild. Die korrekten Projektionsparameter der Kamera müssen so häufig in einem zusätzlichen Kalibrierungsschritt vor der Erkennungsphase erst bestimmt werden (Oikonomidis u. a., 2011c; Breuer, 2005).

[9] Alternative könnten 2D-Modelle (vgl. Bretzner u. a., 2002) oder statistische Modelle sein.

Strukturmodell auf Basis von 3D-Primitiven

Zur Modellierung der Handform wird häufig ein Strukturmodell gewählt, das sich aus einfach zu berechnenden 3D-Primitiven zusammensetzt (Stenger u. a., 2001; Sudderth u. a., 2004; Oikonomidis u. a., 2011c). Dies hat den Vorteil, dass die Primitiven als unabhängige Komponenten mit einfachen Mitteln transformiert und dargestellt werden können. Diese Unabhängigkeit ermöglicht zudem die leichtere Anpassung des Modells an die individuellen Handproportionen des jeweiligen Benutzers (Oikonomidis u. a., 2011c).
Als Primitive eignen sich insbesondere Oberflächenquadriken (Zylinder, Ellipsoide, Kugeln, Kegel usw. s. Abb. 2.20) , da diese sich als einfache analytische Funktion in geschlossener Form beschreiben lassen (vgl. Stenger u. a., 2001) und sich der für die Transformation und Projektion notwendige Aufwand erheblich reduzieren lässt.

Stenger u. a. (2001) nutzen daher in ihrer Lösung ein Strukturmodell auf Basis von unterschiedlichen Quadriken (s. Abb. 2.20), die zusätzlich mehr oder wenig angeschnitten werden, um die ideale Form der Handelemente detaillierter anzunähern. Zur Generierung der, in der Lösung verwendeten Kantenmerkmale, nutzen Stenger u. a. (2001) die Eigenschaft der analytisch beschreibbaren Quadriken aus. So ergibt die Projektion einer Quadrik in die Ebene immer einen Kegelschnitt als Umriss. Dieses Wissen kann dazu genutzt werden, um während der Merkmalsgenerierung direkt die Pixel der Umrisse zu erzeugen, ohne dass eine zusätzliche Extraktion von Kanten aus einer vorher *gerenderten* Modelldarstellung erfolgen muss.



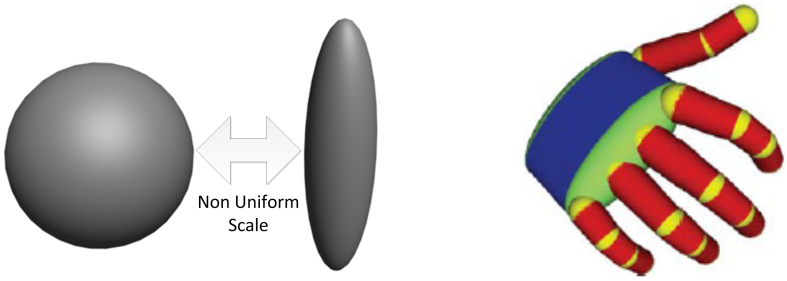
(a) Teile der Hand modelliert durch Quadriken. (b) Transformiertes Quadrikenmodell. (c) Kantenprojektion.

Abbildung 2.20: Handmodell auf Basis von Quadriken (i.A.a. Stenger u. a., 2001)

Das von Stenger u. a. (2001) vorgeschlagene Strukturmodell wird in einigen späteren Arbeiten in angepasster Form aufgegriffen. So nutzen Sudderth u. a. (2004) ebenfalls die Projektionseigenschaften eines aus Quadriken erstellten Strukturmodells zur Generierung der Modellumrisse aus.

Ein Ansatz, der im Gegensatz zu Stenger u. a. (2001) und Sudderth u. a. (2004) den Vergleich der Oberfläche von Modellquadriken mit observierten Merkmalen vorsieht beschreiben Causo

u. a. (2008). Um dabei den Unterschied zwischen der Oberfläche einer realen Hand und der der modellierten Hand möglichst gering zu halten, modellieren Causo u. a. (2008) das Strukturmodell durch eine hohe Anzahl an Oberflächenquadriken (744 Quadriken).
Ein generelles Problem bei der Verwendung vieler unterschiedliche Primitive oder Quadriken wie bei Causo u. a. (2008) besteht allerdings darin, dass während der Posenerkennung jede Klasse von Quadriken durch spezielle Methoden verarbeitet werden muss. Dies kann zu einem zusätzlichen Aufwand führen der sich in der Unterscheidung der einzelnen Arten von Quadriken äußert.
Oikonomidis u. a. (2011c) begegnen diesem Problem indem sie sich die Skalierungseigenschaften von Zylindern und Kugeln zu nutze machen. So lässt sich aus einem Zylinder durch die nicht-uniforme Skalierung, ein elliptischer Zylinder oder ein Kegel formen, aus einer Kugel ein Ellipsoid (s. Abb. 2.21a). Zur näherungsweisen Modellierung der einzelnen Handkomponenten reichen die zwei Basisqaudriken mit entsprechenden Skalierungsfaktoren und einer Beschneidung aus (s. Abb. 2.21b), mit dem Resultat, dass eine wesentlich generellere Verarbeitung der Modellstruktur möglich ist.



(a) Nicht-uniforme Skalierung von Quadriken. Aus einer Kugel wird ein Ellipsoid.

(b) Angepasstes Handmodell (i.A.a Oikonomidis u. a., 2011c).

Abbildung 2.21: Skalierung zur Anpassung einzelner Quadriken an die reale Handform.

Strukturmodell auf Basis eines Polygon Mesh

Die Definition des Strukturmodells auf Basis einzelner Primitiven birgt den Nachteil, dass die reale Form einer Hand je nach Anzahl und Art der Primitiven nur näherungsweise nachempfunden werden kann. Um diesem Nachteil entgegen zu wirken, wird in der Literatur häufig die Verwendung eines einheitlichen *Polygon Mesh* vorgeschlagen (Ying Wu und Huang, 1999; Bray u. a., 2004; Brown und Capson, 2012). Ein Polygonnetz (*Polygon Mesh*), das sich aus einzelnen Scheitelpunkten (*Vertices*) und deren triangulierten Oberflächen (*faces*) zusammensetzt (s. Abb. 2.22), erlaubt dabei eine wesentlich angepasstere, detailliertere Beschreibung der Handoberfläche, wovon man sich in der Praxis meist eine genauere Messung des Anpassungsfehlers erhofft (Bray u. a., 2004).

Die Transformation und Projektion eines *Polygon Mesh*, ist je nach Auflösung mit einem höheren Aufwand verbunden. Da moderne GPUs sich jedoch auf die Verarbeitung polygonbasierter

Objekte spezialisieren und durch ihre *SIMD-Architektur* (*Single Instruction Multiple Data*) eine parallelisierte Berechnung großer Mengen von *Vertices* und *Faces* mit geringen Laufzeiten zulassen, sollte dieser Aspekt nicht so ins Gewicht fallen.

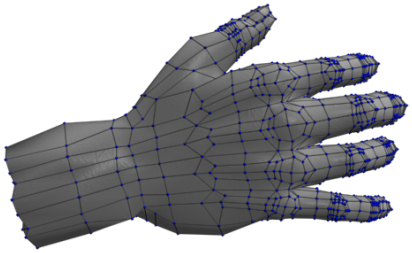


Abbildung 2.22: *Polygon Mesh* als Menge von Scheitelpunkten (*Vertices*) und deren Triangulation (*faces*).

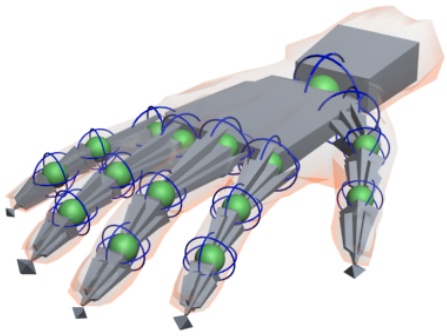
2.5.2 Kinematikmodell

Um eine kontinuierliche Erkennung von Handposen durch eine Anpassung des Modells an die jeweilige Aufnahme zu erreichen, muss, neben der Beschreibung des statischen Strukturmodells, die Modelldynamik in Form eines Kinematikmodells beschrieben werden. Dazu wird im Kinematikmodell die Art und Anzahl der jeweiligen Modellparamter festgehalten. Zudem wird hierin definiert, in welcher Abhängigkeit einzelne Komponenten des Strukturmodells zu den einzelnen Modellparametern stehen und in welcher Weise diese durch die Veränderung eines Parameters transformiert werden.
Da die Verwendung mehrerer stetiger Modellparameter sehr viele mögliche Modellkonfigurationen zur Folge hätte, die keiner realen Pose der Hand entsprechen, ist es zur Vereinfachung der Posenerkennung häufig ratsam, im Sinne einer Dimensionsreduzierung, den Wertebereich der Parameter auf das Nötigste zu beschränken. Das Kinematikmodell mit der Menge der Modellparameter kann so als Synonym für die Bewegungsmöglichkeiten der einzelne *DOF* des realen Handskelletes gelten, so dass die Modellparameter in der Literatur meist ebenfalls als *DOF* bezeichnet werden.

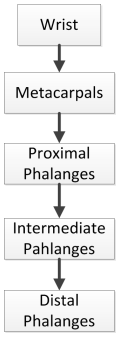
Skeleton Model

Wie bereits angesprochen wird häufig ein kinematisches Modell verwendet, dass die Bewegungsmöglichkeiten der einzelnen Knochen das Handskelletes nachempfindet Bray u. a. (2004); Oikonomidis u. a. (2011b); La Gorce u. a. (2011). So ergibt sich die, über die Modellparameter gesteuerte, Transformation der einzelnen Komponenten, wie bei der realen Hand, aus der mehr oder weniger eingeschränkten lokalen 3D-Rotation um die Verbindungspunkte (Joints)

zwischen den einzelnen Modellkomponenten (s. Abb. 2.23a). Die Anzahl der Joints muss dabei nicht zwingend die tatsächlichen Anzahl realen Handgelenke widerspiegeln.



(a) Skeleton Model als Menge von Knochen und Joints. Die Transformation erfolgt durch lokale 3D-Rotationen um die Joints (hier in Grün) zwischen den einzelnen Komponenten bzw. Knochen.



(b) Hierarchischer Zusammenhang zwischen einzelnen Komponenten in Form einer kinematischen Kette. Die absolute Position einer Komponente ist immer abhängig von der Position seines Vorgängers.

Abbildung 2.23: Skeleton Model und deren Komponentenhierarchie.

Da ein skelettbasiertes Kinematikmodell nur lokale Rotationen um die jeweiligen Joints berücksichtigt, besteht hier implizit ein hierarchischer Zusammenhang zwischen den einzelnen Komponenten des Modells (s. Abb. 2.23b), was sowohl Vor- als auch Nachteile mit sich bringt (Sajjawiso und Kanongchaiyos, 2011).
Ein Vorteil ist, dass die Hierarchie eine einfache Reduzierung der Dimensionalität zulässt. So lassen sich die Rotationswinkel meist durch einfach zu beschreibende *Constraints* auf tatsächlich observierbare Winkel einzelner Komponenten (z.B. keine Kollisionen einzelner Fingerglieder und keine übernatürliche Drehung von Fingergliedern) beschränken (Oikonomidis u. a., 2011b).
Ein wesentlicher Nachteil ist jedoch, dass zur absoluten Beschreibung einer Modellkonfiguration bzw. Modellpose immer ein sequenzieller Durchlauf der Komponentenhierarchie (kinematische Kette) erforderlich ist, was sich vor allem auf das *Tracking* der Parameter auswirkt. So muss ein *Tracking* sich immer auf den, durch die die gesamte Menge der Parameter gegebenen, globalen Zustand beziehen, was die Verwendung einer effizienten parallelisierten Tracking-Architektur, in der jeder Parameter separat verfolgt wird, ausschließt.

Disjoint Model

Als Gegenstück zum *Skeleton Model* haben sich in der Literatur Kinematikmodelle etabliert, die die Parameter nicht mehr als lokale Rotation einzelner Handkomponenten um die einzelnen Joints betrachten, sondern die Bewegung der einzelnen Komponenten unabhängig voneinander

durch die räumliche globale Translation und Rotation modellieren(Ying Wu und Huang, 1999; Sudderth u. a., 2004; Hamer u. a., 2009).
Die einzelnen Modellparamter reflektieren so nicht mehr die realen *DOF* der Hand, erlauben in ihrer Gesamtheit aber eine eindeutige Beschreibung des jeweiligen Handzustands und damit der Handpose (s. Abb. 2.24). Oikonomidis u. a. (2011b) bezeichnen die Verfahren, die einem solchen kinematischen Modell entspringen daher auch als *Disjoint Evidence Methods*.

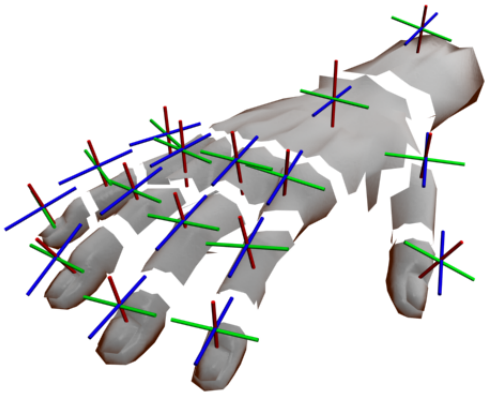


Abbildung 2.24: Disjoint Model. Einzelne Komponenten werden unabhängig voneinander transformiert.

Die Entkopplung der einzelnen Parameter hat dabei den Vorteil, dass einzelne Modellkomponenten, im Gegensatz zu dem *Skeleton Model*, direkt durch die drei Rotationsparameter und die drei Translationsparameter absolut positioniert werden können. Dies führt dazu, dass das *Tracking* der Parameter für jede Komponente unabhängig und in einzelne Teilschritte separiert werden kann. Dadurch lässt sich das *Tracking* im hohen Maße parallelisieren (vgl. Hamer u. a., 2009).

Ein generelles Problem eines *Disjoint Model* ist jedoch, dass durch die unabhängige Transformierbarkeit der einzelnen Modellkomponenten die realen Zusammenhänge der Hand nicht mehr implizit durch das Modell beschrieben werden. So besteht hier prinzipiell die Möglichkeit, dass einzelne Komponenten an beliebige Stellen des Raums verschoben werden können, ohne dass überhaupt eine Verbindung zu den Nachbarkomponenten gewährleistet wird. Für ein parallelisiertes *Parametertracking* bedeutet dies, dass ohne weitere Gegenmaßnahmen die Gefahr besteht, dass Parameterkonstellationen bestimmt werden können, die zwar einen minimalen Fehler liefern allerdings keiner realen Pose entsprechen. So besteht bei *Disjoint evidence methods* der Zwang die strukturellen und dynamischen Zusammenhänge der Hand in Form von *Constraints* explizit zu formulieren (vgl. Sudderth u. a., 2004). Die Verwendung globaler Transformationsparameter anstatt der, wie im *Skeleton Model* verwendeteten, lokalen Rotationen, erschwert dabei meist die Formulierung einzelner Constraints, die sich auf den jeweiligen Wertebereich eines Parameters beziehen (Oikonomidis u. a., 2011b).

Reduzierung der Dimension

Betrachtet man die Kinematik der Hand als durch die DOF beschriebenen uneingeschränkten Rotationen der einzelnen Komponenten um die Verbindungsstellen der einzelnen Knochen des Skelettes, ergibt sich durch die DOF-Anzahl von über 20 ein stetiger DOF-Raum mit sehr hoher Dimensionalität mit einem einheitlichen Wertebereich jedes Parameters zwischen 0° und 360°.

In der Realität ist die Bewegungsmöglichkeit einzelner Komponenten allerdings auf einen be-

stimmten Rotationsbereich beschränkt (z.B. Knicken der Fingerspitze), wodurch sich die Bereichsgrenzen der einzelnen Dimensionen des realen DOF-Raums durch die Grenzen des Machbaren ergeben. Zudem ist die Kinematik der Hand durch eine Hierarchie gegeben (s. Abschnitt 2.5.2), in der der natürliche Wertebereich der parametrisierten Rotation einer Handkomponente direkt von der jeweiligen Stellung der vorangehenden Komponente abhängt [10]

Die Berücksichtigung dieser kinematischen Einschränkungen kann dabei zur effektiven Reduzierung der Problemdimensionalität genutzt werden, was einige Vorteile mit sich bringt. So müssen einerseits nur Parameterwerte innerhalb des gültigen Wertebereichs eines *DOF* überprüft werden, was die Parametersuche prinzipiell beschleunigt. Andererseits wird die Suche nach einer posenbeschreibenden Parameterkonstellation auf die Menge der real möglichen Zustände einer Hand begrenzt, wodurch keine unrealistischen Werte in Form von Fehldetektionen als Ergebnis geliefert werden.

In der Praxis erfolgt eine Reduzierung der Dimension meist durch explizite Formulierung von *Constraints* (Sudderth u. a., 2004; Hamer u. a., 2009; Oikonomidis u. a., 2011a). Die Verwendung der Constraints reicht dabei von der einfachen Überprüfung des Wertebereichs (s. Abb. 2.25) durch *Domain Constraints* (Oikonomidis u. a., 2011b) bis hin zu schwerer zu formulierenden dynamischen *Constraints* (Sudderth u. a., 2004; Hamer u. a., 2009), die eine zeitliche Kontinuität zur Reduzierung der hierarchisch abhängigen Wertebereiche mit einbeziehen.

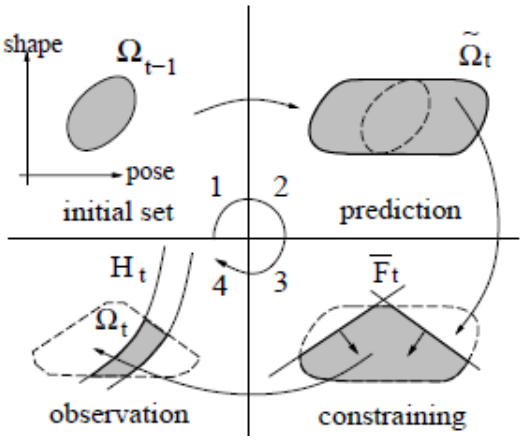


Abbildung 2.25: Dimensionsreduzierung durch Constraints nach Shimada u. a. (2001). Zu Beginn eines Aufnahmeschritts stehen die ermittelten Parameter des vorherigen Schritts fest (Quadrant 1). Durch das Tracking werden dann Vorhersagen um den Bereich der Ausgangsparameter gemacht (Quadrant 2). Die Vorhersagen können durch die Auswertung der Constraints auf die möglichen Parameterkonstellationen beschränkt werden (Quadrant 3). Schließlich kann die reduzierte Anzahl der Parametervorhersagen mit der Observation verglichen werden (Quadrant 4).

[10] Bspw. ist beim ausgestreckten Finger kaum eine Biegung der Fingerspitze möglich, bei geknickten Finger dagegen schon.

Eine Schwierigkeit bei der Formulierung einzelner Constraints besteht darin, die Bewegung der Hand möglichst exakt einzuschätzen und dabei robust gegenüber den natürlichen Variationen der Handbewegung zu sein. Um möglichst generelle Aussagen über die einzelnen Handbewegungen treffen zu können, werden daher in einigen Lösungen die dynamischen Zusammenhänge der Hand in einem vorherigen Schritt durch ein trainingbasiertes Verfahren erlernt (vgl. Stenger, 2004).

2.5.3 Observationsmodell

Die wesentliche Anforderung an ein Observationsmodell ist die, einen schnellen Vergleich , unter Berücksichtigung des jeweiligen Strukturmodells, zwischen extrahierten Merkmalen und generierten Merkmalen zu realisieren. Dabei sollte zudem eine ausreichend genaue Messung des Fehlers gewährleistet werden. Wird eine einzelne Farbkamera zur Aufnahme verwendet, werden in der Regel kantenbasierte Merkmale als Basis des Vergleichs gewählt, da diese weniger von Mehrdeutigkeiten betroffen sind als bspw. die Handkontur. Einen genaueren Vergleich liefern dabei nur Merkmale die die Topologie der Handoberfläche mit einbeziehen. Für die einfache und effiziente Bestimmung dieser Oberflächenmerkmale ist jedoch meistens eine dreidimensionale Wahrnehmung der Hand, z.B. durch die Aufnahme eines Tiefensensors, erforderlich.

Observationsmodell auf Basis von Kanten

Zur schnellen Bestimmung des Fehlers zwischen Modell und Observation anhand projizierter Modellkanten und den extrahierten Kanten des Eingabebildes nutzen Stenger u. a. (2001) eine Methode, die ähnlich wie die *Chamfer Distanz Transform*, die Entfernung zwischen zwei korrespondierenden Pixeln misst. Dabei wird von jedem Modellkantenpixel ausgehend in der Richtung der Kantennormalen der nächstliegende Kantenpixel des Eingabebildes gesucht. Die Extraktion eines Eingabekantenpixels erfolgt dabei dynamisch, innerhalb der Suche, so dass es weder einer Extraktion aller Kantenpixel des Eingabebildes noch einer vorangehenden kompletten Distanztransformation bedarf. Aus den Entfernungen der einzelnen Modellpixel zu ihren korrespondierenden Kantenpixel im Eingabebild wird dann der gesamte Fehler berechnet.

Die dynamische Suche nach einem korrespondierenden Eingabepixel auf Basis eines Modellpixels hat jedoch den Nachteil, dass diese durch ihre sequenzielle Struktur eher nicht auf einer GPU umgesetzt werden kann. (Oikonomidis u. a., 2011c) wählen daher einen anderen Ansatz als Stenger u. a. (2001) und nutzen eine komplette Distanztransformation des Eingabebildes für ihren *GPU*-basierten Vergleich. Um eine Messung des Fehlers möglichst schnell für viele hypothetische Modellkonfigurationen durchführen zu können, wird in dem Lösungsvorschlag ein *Tile-basierter* Ansatz verfolgt. Dabei werden die Kanten das Strukturmodells anhand der angenommenen Parameterkonstellation mehrfach in definierte Bereiche (*Tiles*) eines großen

Referenzbildes *gerendert*. Die Bestimmung des Fehlers eines *Tiles* anhand des distanztransformiertem Eingabebild erfolgt schließlich durch einen angepassten *Pixelshader*.

Observationsmodell auf Basis von Oberflächenmerkmalen

Einen Ansatz vor, der den Vergleich der Oberflächenbeschaffenheit eines, aus einer hohen Anzahl an Oberflächenquadriken bestehendem, Modells mit einer über mehrere Kameras wahrgenommen Hand, vorsieht, stellen Causo u. a. (2008) vor. Als Basis des Vergleichs dient hierbei ein aus den Eingabebildern der einzelnen Kameras abgeleitetes Voxel-Model, das eine nahezu exakte Wahrnehmung des Handvolumens darstellt (s. Abb. 2.26). Zur Messung des Fehlers wird dann für den Mittelpunkt der Oberfläche jeder Modellquadrik der naheliegendste Voxel aus dem observierten Modell bestimmt, wobei der Abstand zwischen dem jeweiligen Mittelpunkt und Voxel in Form der Manhattan Distanz als Metrik für den Fehler gilt.

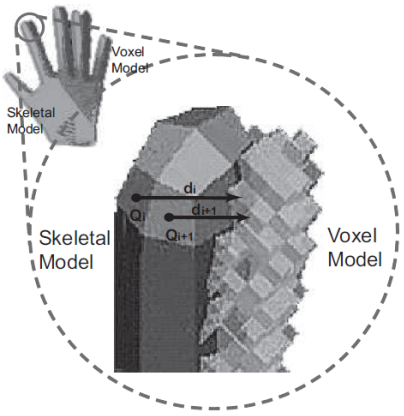


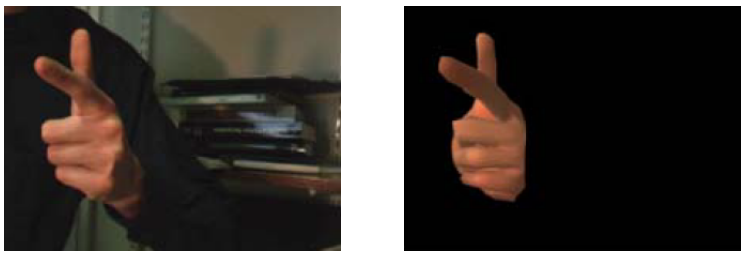
Abbildung 2.26: Distanz zwischen Modellquadrik und *Voxel* der Observation als Fehlermetrik (i.A.a. Causo u. a., 2008)

Eine Schwierigkeit in der vorgeschlagenen Lösung könnte jedoch in der kontinuierlichen Ableitung eines Voxel-Modells bestehen. So müssten für die Bildung des Modells Übereinstimmungen in sämtlichen observierten Handregionen ermittelt werden und zu einem Gesamteindruck in Form des Voxel-Modells zusammengefasst werden. Dies kann einen nicht zu vernachlässigen Aufwand bedeuten und zu hohen Laufzeiten führen.

Um die Messung des Fehlers zwischen der Modellanpassung und der über einen Tiefensensor aufgenommenen Handoberfläche möglichst effizient zu gestalten, beschränken sich Bray u. a. (2004) auf die Auswertung markanter Punkte auf der Modelloberfläche. Da die Messbarkeit der Punkte von der jeweiligen Pose bzw. Modellkonfiguration abhängt, werden diese über einen zusätzlichen stochastischen Prozess, der an ein Tracking von Merkmalen erinnert (s. Abschnitt 2.3.1), dynamisch in jedem Aufnahmeschritt neu verteilt. Die Messung des *Per-Pixel-Fehlers* erfolgt dabei durch die Projektion der Modellpunkte in die Ebene des Eingabebildes und dem direkten Vergleich mit den dort befindlichen Tiefenpixeln. Um diesen auf die einzelnen Punkte beschränkten Vergleich etwas objektiver zu gestalten erfolgt hier zudem eine Gewichtung des pro Pixel bestimmten Unterschieds durch den jeweiligen Gradienten an der ermittelte Bildposition des Modellpunktes.
Diese Beschränkung auf einzelne markante Modellpunkte während der Fehlerauswertung führt jedoch dazu, dass hier weniger Informationen betrachtet werden, sodass sich das Risiko für Fehldetektionen aufgrund von Mehrdeutigkeiten und Selbstverdeckung erhöht.

Oikonomidis u. a. (2011b) dagegen sehen den direkten Vergleich zwischen den einzelnen Pixeln eines aufgenommenen Tiefenbildes und einer projizierten Hand als Basis ihres Observationsmodells. Der pixelbasierte Vergleich eignet sich dabei insbesondere zur GPU-basierten Implementierung. Die Vergleichswerte der einzelnen Pixel werden schließlich über eine spezielle Fehlerfunktion gemittelt und zu einem skalarem Wert zusammengefasst.

Eine Verfahren das versucht die Handoberfläche ohne Tiefeninformationen messbar zu machen, stellen La Gorce u. a. (2011) vor. Ziel ist dabei, einen pixelbasierten Vergleich zwischen dem Aufnahmebild einer monocolaren Kamera und einer dynamisch angepassten Textur des Modells zu realisieren (s. Abb. 2.27), der dann zur Bestimmung des Fehlers genutzt wird. Eine grundsätzliche Schwierigkeit ist dabei, eine möglichst realistische Texturierung des Modells zu erreichen. Zudem kann die Textur über den Zeitraum einer Posenerkennung nicht konstant bleiben, da unterschiedliche Konfiguration der Hand und Beleuchtungseinflüsse der Umgebung zu unterschiedlichen Schattierungen der observierten Hand führen. La Gorce u. a. (2011) berücksichtigen dies, in dem sie ein explizites Texturmodell einführen, das eine dynamische Anpassung der Texturschattierung über einzelne Parameter erlaubt. Die einzelnen Texturparameter werden dann zusätzlich zu den Parametern des Strukturmodells in einem Tracking berücksichtigt, sodass in jedem Schritt der Aufnahme eine passende Textur zum direkten Vergleich der Pixel des Eingabebildes bereit steht.
Für die reine geometrische Deutung der Handpose erscheint dieses Verfahren eher umständlich und zu ungenau, da es durch die Einführung zusätzlicher Parameter prinzipiell zu einer Erhöhung der Dimensionalität führt und es fraglich ist ob die Texturparameter durch sich schnell verändernde Lichtverhältnisse (z.B. TV, Bildschirm) noch eindeutig bestimmbar sind. Ist jedoch die Erkennung der Textur das vordringliche Ziel, erscheint die vorgestellte Lösung als ein probates Mittel.



(a) Observiertes Bild.

(b) Ergebnis der Posenerkennung mit angepasster Textur.

Abbildung 2.27: Messung des Fehlers auf Basis einer Modelltextur.

2.5.4 Parameter Tracking

Ein erster Ansatz der Parameterbestimmung könnte darin bestehen den Fehler zwischen Modell und dem jeweiligen Eingabebild direkt ohne Berücksichtigung der Dynamik durch ein entsprechenden Optimierungsverfahren (z.B. Gradientenabstieg) zu minimieren. Da der Modellparameterraum trotz Reduzierung der Dimensionalität meist sehr groß und nichtlinear ist, und da es ohne Berücksichtigung der Dynamik keinen optimalen Ansatzpunkt für eine Parametersuche innerhalb der Fehlerfunktion gibt, ist das Risiko hier sehr groß das nicht das globale Minimum sondern ein nicht korrektes lokales Minimum der Fehlerfunktion bestimmt wird (s. Abb. 2.28).

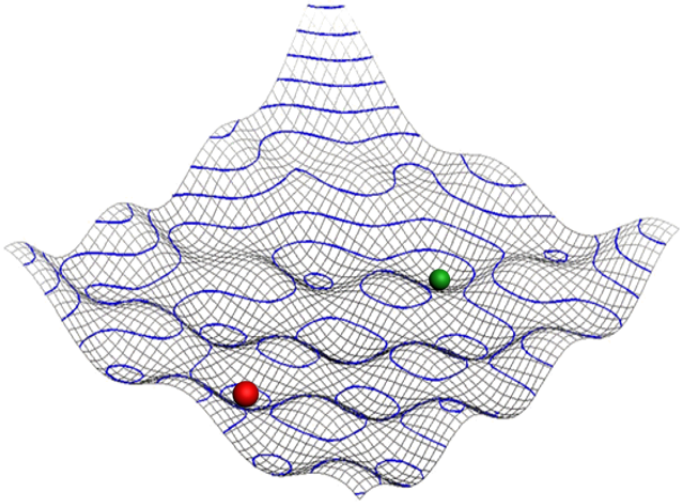


Abbildung 2.28: Parameterbestimmung als Optimierung der Fehlerfunktion. Ziel globales Optimum (grün). Fehlerhafte Erkennung durch Bestimmung eines lokalen Optimums (rot).

Um diesen Umstand zu vermeiden und dadurch eine genauere Posenerkennung zu gewährleisten muss daher, wie am Anfang des Abschnitts 2.5 angesprochen, meist die zeitlichen Kontinuität durch ein Tracking der Parameter mit einbezogen werden. So unterscheiden sich die zu bestimmenden Parameter im aktuellen Aufnahmeschritt meist nur geringfügig von den zuvor ermittelten Parameterwerten, sodass die bereits gewonnenen Erkenntnisse als optimaler Ansatzpunkt für eine Parametersuche gelten können und das Risiko in einem lokalen Minimum zu landen reduziert wird.

Ein Tracking lässt sich daher formal als rekursive Funktion beschreiben, die in Abhängigkeit der ‚in einem vorherigen Schritt gemessenen, Parameter $\vec{x_{t-1}}$ eine oder mehrere Vorhersagen $\vec{x_t}$ zur Parameterkonstellation im aktuellen Aufnahmeschritt als Ergebnis liefert.

$$\vec{x_t} = a(\vec{x_{t-1}}) \times \vec{x_{t-1}} \tag{2.3}$$

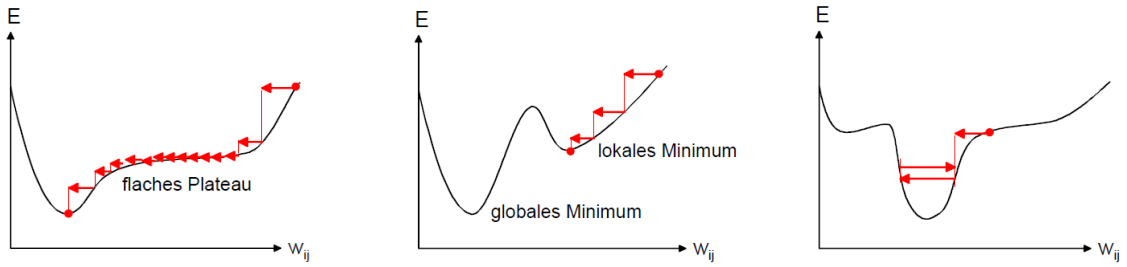
Die Funktion $a(\vec{x_{t-1}})$ kann dabei als Aktualisierungstherm aufgefasst werden, der auf Basis bisheriger Erkenntnisse eine gegebene Parameterkonstellation neu bewertet.

Grundsätzlich lassen sich laut Erol u. a. (2007) die bisher verwendeten Tracking-Verfahren in die zwei Kategorien *Single Hypothesis Tracking* bzw. deterministische Verfahren und *Multiple Hypothesis Tracking* unterteilen, wobei erstere zu jedem Aufnahmeschritt nur eine einzelne optimale Vorhersage über die Verteilung der Parameter im nächsten Bild treffen.

Single Hypothesis Tracking

Gradienten-basierte Verfahren:
Als deterministischen Optimierungsverfahren werden häufig Methode des Gradientenabstiegs verwendet. Hierbei wird ausgehend von einer definierten Position innerhalb der Funktion der Gradient bestimmt, um darauf aufbauend einen Schritt in Richtung des steilsten Funktionsabstiegs zu tätigen. An der durch den Schritt beschriebenen aktualisierten Position wird dann wiederum der Gradient bestimmt und erneut ein entsprechender Schritt durchgeführt, usw. So konvergiert das Verfahren schrittweise zu einem Minimum der Funktion.
Zum Tracking der einzelnen Modellparameter kann die sequenzielle Struktur des Gradientenabstiegs ausgenutzt werden, um in jedem Aufnahmeschritt die letzte Parameterkonstellation anhand deren verursachter Fehlergradienten durch einen verbesserten Vorschlag für die Parameter (*Update*) zu ersetzen. Dieser verbesserte Vorschlag wird im folgenden Aufnahmeschritt erneut anhand der Gradienten bewertet und wieder aktualisiert. Eine Lösung, die einen solchen Ansatz verfolgt, beschreiben Ren u. a. (2011), wobei diese eine dynamische Variante des *Gauss-Newton Algorithmus* zur Optimierung einsetzen.

Eine wesentliche Größe eines gradientenbasierenden Optimierungsverfahrens ist die Weite des Schrittes, der in Richtung des Gradienten gegangen wird (Meisel, 2012). Basiert diese Schrittweite ausschließlich auf der jeweiligen Gradientenstärke kann dies, je nach Beschaffenheit der Fehlerfunktion, zu einer langsamen Konvergenz oder zu Singularitäten führen (s. Abb. 2.29).



(a) Flache Plateaus führen zu einer langsamen Konvergenz.

(b) Es kann gegen ein lokales Minimum konvergiert werden.

(c) Bei steilen Gradienten ist eine Oszilation möglich.

Abbildung 2.29: Probleme bei Gradientenstärke als Schrittweite (i.A.a. Meisel, 2012).

Bray u. a. (2004) nutzen daher für ihr Modell des Gradientenabstiegs ein Verfahren namens *Stochasitic Meta Descent* (SMD), das eine dynamische Anpassung eines zusätzlichen Schrittweitenfaktors vorsieht. So wird dieser in jedem Aufnahmeschritt genau wie die konkrete Parameterkonstellation aktualisiert. Zur Aktualisierung des Schrittweitenfaktors bzw. Schrittweitenvektors werden dabei statistische Methoden eingesetzt, die die Zusammenhänge zwischen den Ergebnissen der Vergangenheit und dem aktuellen Ergebnis analysieren und in dem Faktor zusammenfassen. So kann der Schrittweitenfaktor als Autokorrelationsfaktor für die Aufnahmesequenz (bzw. des Signals) gelten. Der so bestimmte Schrittweitenfaktor kann dann in Kombination mit den ermittelten Gradienten der Fehlerfunktion zur optimaleren Aktualisierung der Parameterkonstellation verwendet werden.

Ein Problem bei der Verwendung gradientenbasierter Verfahren zum Tracking eines Handmodells ist, dass diese immer auf den partiellen Ableitungen 1. Ordnung (manchmal auch 2. Ordnung) einer nicht genau bekannten sehr störungsanfälligen Fehlerfunktion basieren. Mehrdeutigkeiten und andere Defizite der Fehlerfunktion können daher leicht dazu führen, dass falsche Gradienten bestimmt werden oder einzelne lokale Minima verfolgt werden, was in beiden Fällen ein divergierendes Tracking zur Folge hätte. Ein zusätzliches Problem der gradientenbasierten Verfahren ist, das diese meist eine Berechnung der partiellen Ableitungen in Form einer Jacobi-Matrix vorsehen. Die Erstellung einer Jacobi-Matrix erfordert dabei immer einen, von der jeweiligen Parameterzahl abhängigen, quadratischen Aufwand. Da die Anzahl der Modellparameter meist mehr als 20 beträgt und die Berechnung der partiellen Ableitung immer mindestens zwei Messungen des Fehlers zwischen Modell und Observation bedeutet, ist der Erstellungsaufwand für eine Jacobi-Matrix nicht unerheblich und kann zu zeitkritischem Verhalten führen.

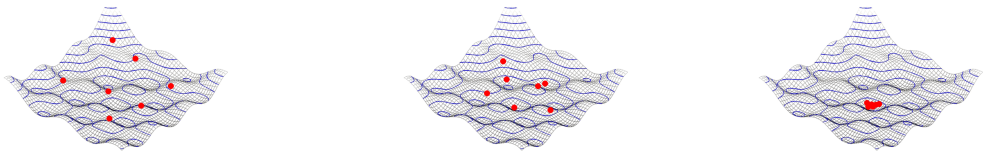
Kalman-Filter:
Die Nachteile der gradientenbasierten Verfahren motiviert häufig dazu, *Tracking*-Algorithmen einzusetzen, die unabhängiger von der jeweiligen Fehlerfunktion und dadurch robuster gegenüber einzelnen Messfehlern sind. Dabei lassen sich diese meist in die Klasse der stochastischen Filter einordnen, die in der Systemtheorie zur rekursiven Abschätzung eines folgenden Systemzustands genutzt werden.
Ein sehr prominenter Vertreter dieser Verfahren ist der Kalman-Filter, der sich primär dadurch auszeichnet, dass die stetig wachsende Messreihe aus den Mittelwerten der bereits bestimmten Parameter der Vergangenheit, in Form einer Linearkombination und unter Berücksichtigung der Varianz zusammengefasst in jedem Aufnahmeschritt zur Verfügung steht. Darauf aufbauend kann eine optimale Hypothese für die Parameterverteilung im folgenden Aufnahmeschritt abgeleitet werden. Die Richtung der Optimierung wird daher nur durch die zustandsabhängige Gewichtung vergangener Ergebnisse[11] vorgegeben, ohne dass es einer Auswertung von Gradienten der Fehlerfunktion bedarf. Da sich der Kalman-Filter jedoch nur für das Tracking linearer Systeme[12] verwenden lässt, die Parameterbestimmung jedoch ein stark nichtlineares System

[11] und zusätzlichen Rauschanteil
[12] Ein Lineares System liegt nach systemtheoretischen Erkenntnissen dann vor, wenn ein Ausgangssignal proportional zum Eingangssignal ist, und beide Eingabesignale unkorreliert sind (vgl.)http://de.wikipedia.org/wiki/Linearität

darstellt (Stenger, 2004), muss hier auf die Erweiterungen in Form des *Extended Kalman Filter* (*EKF*) oder des *Unscented Kalman Filter* (UKF) zurückgegriffen werden. Beide Varianten sehen dabei eine Linearisierung des Systems vor. Stenger (2004), der einen UKF zum Modellparameter Tracking einsetzt, betont jedoch dass der UKF eine bessere Performance erzielt.

Particle Swarm Optimization:
Ein Problem bei den vorher beschriebenen Verfahren ist, dass diese nur eine lokale Optimierung durchführen. So wird unabhängig davon welche Richtung der Optimierung durch die Gradienten oder die Ergebnishistorie vorgegeben wird, eine einzelne Hypothese erzeugt und als beste Lösung in einem *Tracking-Schritt* betrachtet. Aufgrund von Messfehlern, Mehrdeutigkeiten oder anderen Störeinflüssen können jedoch falsche Hypothesen gebildet werden, die dann von Aufnahmeschritt zu Aufnahmeschritt weitergereicht werden und zu einem divergenten *Tracking* führen.

Um diesem Problem zu entgegnen, verwenden Oikonomidis u. a. (2011c) das relativ neue Verfahren der *Particle Swarm Optimization*. Hierbei wird eine Menge von Partikeln erzeugt und zufällig um den Bereich der im vorherigen Aufnahmeschritt bestimmten Parameterkonstellation im Parameterraum verteilt (s. Abb. 2.30). Für jeden dieser einzelnen Partikel wird nun eine Optimierung, bei der die einzelnen Partikel interagieren, über die Fehlerfunktion durchgeführt, wobei immer der Partikel der den geringsten bisherigen Fehler geliefert hat festgehalten wird. Durch die Interaktion der Partikel und dem global festgehalten minimalen Fehler kann der einzelne Partikel den globalen Zustand besser einschätzen und leichter zu einem globalen Optimum konvergieren. Nach dem erreichen eines Abbruchkriteriums, wird die Parameterkonstellation des Partikels mit dem geringsten Fehler als Ergebnis der Posenerkennung geliefert.



(a) (b) (c)

Abbildung 2.30: Visualiseierte Minimierungssequenz der *Particle Swarm Optimization* (Quelle: www.itm.uni-stuttgart.de). Die Optimierung erfolgt über den gesamten Parameterraum.

Durch die verteilte Optimierung ermöglicht die *Particle Swarm Optimization* eine genauere Einschätzung der Parameterkonstellation in einem Aufnahmeschritt und sollte daher generell eine robustere Erkennung ermöglichen, allerdings ist für die Auswertung der einzelnen Partikel ein gewisser Aufwand erforderlich. Die Auswertung lässt sich aber im höchsten Maße parralelisieren und eignet sich daher insbesondere für die Implementierung auf der GPU (Oikonomidis u. a., 2011b).

Multiple Hypothesis Tracking

Ein generelles Problem bei dem *Single Hypothesis Tracking* besteht darin, dass zu jeder Zeit nur eine Hypothese formuliert wird. Tracking-Fehler, die durch eine Fehleinschätzung des Folgezustands, durch lokale Minima, Singularitäten, Störeinflüsse, der Unvollkommenheit der Fehlerfunktion oder zu komplexe Bewegungsabläufe der Hand immer wieder auftreten können, können so zum unweigerlichen Abbruch des Tracking führen (Erol u. a., 2007). Als Lösung können hier *Multiple Hypothesis Tracking* Verfahren gelten, die jeweils mehrere Hypothesen während der Aufnahme propagieren. Tritt jetzt ein Tracking-Fehler bei einer der Hypothesen auf, kann immer noch auf die restlichen Hypothesen zurück gegriffen werden. Ein Aspekt der das Tracking auch über einen sehr langen Zeitraum sehr robust macht.

Ansätze wie *Grid-basierte Verfahren* (Stenger u. a., 2006) kommen zum nicht deterministischen *Multiple Hypothesis Tracking* in Frage. Allerdings sehen diese meist eine Diskretisierung des Zustandsraums bzw. Parameterraums vor, was einer kontinuierlichen Bestimmung der Modellparameter widerspricht.
Ein prominenter Vertreter des *Multiple Hypothesis Tracking*, der ohne Diskretisierung des Zustandsraums auskommt, ist der Partikelfilter bzw. die *Sequenzielle Monte-Carlo-Methode* (*SMC*) (Brown und Capson, 2012). Dieser stochastische Filter basiert dabei auf einer rekursiven Implementierung des *Bayesschen-Filters* und fällt damit ebenfalls in die Kategorie des stochastischen Tracking. Basis des Verfahrens ist nun wie bei der *Particle Swarm Optimazation* die Erzeugung einer diskreten Anzahl von Partikeln, die sich über den Parameterraum verteilen, und daher jeweils eine konkrete Parameterkonstellation repräsentieren. Jeder Partikel wird dabei zusätzlich durch einen Faktor gewichtet der die Wahrscheinlichkeit (posteriori) für die Zugehörigkeit zu einer aktuellen Observation angibt. Durch die Menge aller Partikel wird so eine Approximation der Wahrscheinlickeitsdichte erreicht, wobei die zu Grunde liegende Verteilung auch multimodal sein kann (mehrere *Extrema*), was sich insbesondere zur Bewertung von Mehrdeutigkeiten eignet.
Das Filtern der Partikel erfolgt in jedem Aufnahmeschritt in zwei Phasen. In der ersten Aktualisierungsphase werden die einzelnen Gewichte der Partikel durch die Messung des Fehlers zwischen der, durch die Partikel vertretenen, Modellkonfiguration und der Observation bestimmt. Die Partikel mit den höchsten Gewichten können dabei als Lösungen für den jeweiligen Aufnahmeschritt gelten. In der zweiten Phase können die Partikel mit den kleinsten Gewichten (kleinsten Wahrscheinlichkeiten) im Sinne eines dynamischen Filters verworfen werden und durch neue Partikel die sich in der Nähe der optimalen Lösungen befinden ersetzt werden. Die so gefilterten Partikel werden dann als Hypothesen zum nächsten Tracking-Schritt propagiert wodurch eine sequenzielle Konvergenz erreicht wird (s. Abb. 2.31).

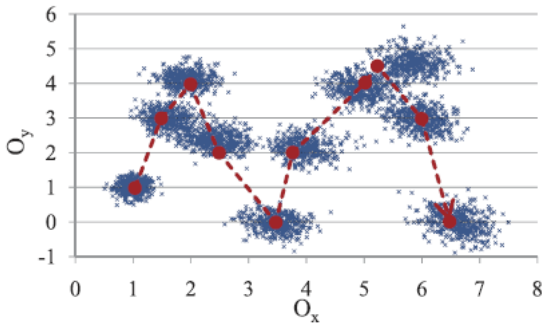


Abbildung 2.31: Konvergenz des Partikelfilter (Brown und Capson, 2012). Die Partikelwolke folgt den Beobachtungen.

Die Bestimmung der Gewichte und anschließende Bewertung ist allerdings genau wie bei der *Particle Swarm Optimazation* durch einen gewissen Grundaufwand gekennzeichnet, der sich vor allem durch die Fehlermessung für jeden einzelnen Partikel bemerkbar macht. Diese Auswertung der Partikel lässt sich aber ebenfalls in hohem Maße parallelisieren. In diesem Zusammenhang stellen Brown und Capson (2012) ihre *GPU-basierte* Implementierung des Partikel-Filters vor, bei der die Auswertung einer großen Menge von Partikeln durch einen *Tile-basiertem* Rendering-Ansatz beschleunigt wird.
Ein Problem des Partikel-Filters ist, dass hier, um eine gute Approximation der Wahrscheinlichkeitsdichte und dadurch eine genaue Posenerkennung zu erreichen, eine sehr hohe Anzahl von Partikeln nötig ist. Die Anzahl der benötigten Partikel hängt dabei exponentiell von der jeweiligen Parameterzahl ab (Brown und Capson, 2012). Für das hochdimensionale Problem des Hand-Tracking bedeutet dies, dass eine sehr hohe Anzahl von Partikeln nötig wird. Diese Menge kann dabei trotz parallelisierter Verarbeitung zu Laufzeitproblemen führen. Ein Modell, in dem einzelne Partikel interagieren, wie die *Particle Swarm Optimazation*, könnte hier zur effektiven Reduzierung der benötigten Partikelanzahl genutzt werden und dadurch einen *Performance-Vorteil* erzielen.

2.5.5 Initialisierung

Da das Tracking der Modellparameter einen rekursiven Zusammenhang zwischen den aktuellen Ergebnissen und den Ergebnissen der Vergangenheit darstellt, muss zu Beginn eines Tracking eine Initialisierung erfolgen. Diese Initialisierung bedeutet, dass die Werte der einzelnen Parameter zu diesem Zeitpunkt mit einer von dem jeweiligen Tracking-Algorithmus vorgegebenen Genauigkeit bestimmt wurden.

Ein Verfahren bei dem es keiner expliziten Initialisierung bedarf, ist das von Stenger u. a. (2001), da das Tracking hier den gesamten Zustandsraum (Parameterraum) zur Bestimmung von Hypothesen mit einbezieht, so allerdings auch nur diskretisierte Antworten liefern kann.

Eine Möglichkeit der Initialisierung, die dem modellbasierten Ansatz treu bleibt, stellt die Ver-

wendung des *Iterative Closesd Point* (*ICP*) Algorithmus dar (Breuer, 2005). Der *ICP* spezialisiert sich dabei auf die geometrische Anpassung zweier Punktwolken, die sowohl aus zweidimensionalen, als auch aus dreidimensionale Koordinaten bestehen können. Häufiger wird jedoch die dreidimensionale Repräsentation der Punktwolken gewählt, da diese weniger anfällig für Mehrdeutigkeiten ist. Die Herleitung der 3D-Punktwolke erfordert allerdings eine Tiefenwahrnehmung. Die Ausführung des *ICP* ist dadurch gekennzeichnet, dass die Punkte der, aus der Aufnahme extrahierten Wolke, und die Punkte der Modellwolke sequenziell verglichen werden. Dabei werden anhand der Richtung und der jeweiligen Abstände korrespondierende Punktepaare gefunden, deren Abstand schließlich minimiert wird.
Der *ICP* liefert jedoch einen hohen quadratischen Aufwand, der bei einer großen Anzahl von Punkten sehr schnell zu langen Laufzeiten führen kann.

Eine effiziente Initialisierung impliziert daher die Verwendung eines *Single View Pose Estimation* Verfahrens (Erol u. a., 2007). In der Literatur wird allerdings meist nur rudimentär auf das Problem der Initialisierung und das Zusammenspiel zwischen einer *Single View Pose Estimation* und einem *Model-based Tracking* eingegangen. So sehen einige Verfahren zur Vereinfachung des Initialisierungsproblems nur eine vorab definierte Parameterkonstellation vor (Initialisierungspose). Zum Starten des Tracking-Vorgangs obliegt dem Benutzer die Aufgabe, die Hand an eine definierte Position im Bild zu verschieben und möglichst exakt diese Initialisierungspose nachzuahmen.

2.5.6 Model-based Tracking Fazit

Die modellbasierte Handposenerkennung bietet den Vorteil, dass diese sich meist einfacher und intuitiver realisieren lässt und dabei eine (nahezu) stetige und kontinuierliche Bestimmung der DOF ermöglicht. Die Verbindung mit einem robusten Tracking erlaubt dann auch einen effektiveren Umgang mit Selbstverdeckung, Mehrdeutigkeiten und zusätzlichen Störeinflüssen, was zu einer generelleren Steigerung Genauigkeit einer Posenerkennung führt.

Im direkten Vergleich mit den exempelbasierten Methoden der *Single View Pose Estimation* lässt sich jedoch festhalten, dass sich der Aufwand von der offline Phase zur online Phase hin verschiebt. So muss zwar keine hohe Anzahl an Exempel aufbereitet und gespeichert werden. Allerdings muss dies durch eine Vielzahl von Messungen, Vergleichen, Transformationen einzelner Pixel, Partikel oder 3D-Primitive zur Laufzeit kompensiert werden. In der Praxis führt dieser Umstand meist dazu, dass modellbasierte Verfahren eher als langsam eingestuft und Erkennungsraten in Echtzeit (> 60 Hz) eher selten zu finden sind.

In dieser Arbeit wird jedoch davon ausgegangen, dass durch die weitere Entwicklung paralleler Hardware-Architekturen und die entsprechende Anpassung der verwendeten Algorithmen, in der Zukunft höhere Erkennungsraten durch modellbasierte Verfahren erzielt werden können.

2.6 Zusammenfassung

Das ideale Ziel einer visuellen Handposenerkennung kann so formuliert werden, dass die einzelnen Bewegungsparameter (*Degree of Freedom* (*DOF*)) der Hand aus der unkorrelierten Menge der Pixel eines Eingabebildes in jedem Aufnahmeschritt wertmäßig bestimmt werden können s. Abschn. 2.1). Dies wird in der Praxis jedoch durch die Umstände der Selbstverdeckung, der Mehrdeutigkeit und der hohen Dimension des DOF-Raums (s. Abschn. 2.1.1) erheblich erschwert.

Um eine Handpose aus der unkorrelierten Menge von Pixeln erkennen zu können, bedarf es häufig einer Vorverarbeitung des jeweiligen Eingabebildes (s. Abschn. 2.2). Diese ist vor allem durch die Schritte der Segmentierung (s. Abschn. 2.2.1) und der darauf aufbauenden Merkmalsextraktion (s. Abschn. 2.2.2) gekennzeichnet.

Zur Posenbestimmung haben sich in der Praxis laut Erol u. a. (2007) zwei unterscheidbare Ansätze etabliert. Die *Partial Hand Pose Estimation* und die *Full DOF Hand Pose Estimation*. Die *Full DOF Hand Pose Estimation* lässt sich weiterhin in *Single View Pose Estimation*- und *Modelbased Pose Tracking*-Ansätze unterteilen.

Um den Aufwand, den eine kontinuierliche Bestimmung der kompletten Handpose (*Full DOF Hand Pose Estimation*) mit sich bringt, zu reduzieren, beschränken sich Ansätze, die in den Bereich der *Partial Hand Pose Estimation* fallen, auf die Erkennung einzelner Handteile oder einzelner kompletter Handposen (s. Abschn. 2.3). Die Erkennungsergebnisse dienen dann häufig als Eingabe für eine anschließende Gestenerkennung. Als Basis der partiellen Erkennung kann die eindeutige Detektion einer Hand im Aufnahmebild angesehen werden, die eine Verfeinerung der bereits extrahierten Merkmale zu weiteren Merkmalen mit möglichst hohem Informationsgehalt (*High-Level Features*), wie z.B. Fingerspitzen erfordert.
Der wesentliche Nachteil der *Partial Hand Pose Estimation* ist, dass sich die Posenerkennung auf ein Anwendungsgebiet spezialisiert, eine generelle korrekte Bewertung der wahrgenommen Hand aber nicht gewährleistet wird. So obliegt es dem Benutzer die Anforderungen, die eine korrekte Posenbestimmung erlauben, zu erfüllen.

Um eine komplette nahezu generelle Erkennung der Handpose im Sinne einer *Full DOF Hand Pose Estimation* möglichst schnell in einem einzelnen Bild zu erreichen, werden Verfahren benötigt die über die Fähigkeiten einer *Partial Hand Pose Estimation* hinausgehen. Dies führt zur *Single View Pose Estimation* (s. Abschn. 2.4). Hier wird versucht, eine direkte möglichst allgemeingültige Zuordnung zwischen den extrahierten Merkmalen und den DOF zu erreichen. Diese Zuordnung wird dabei meist ähnlich wie die partielle Posenklassifizierung durch exempelbasierte Verfahren realisiert, mit dem kleinen Unterschied, dass die Anzahl der benötigten Exempel um einiges höher sein muss, um eine annähernd generelle Erkennung zu ermöglichen. Die Verfahren aus dem Bereich der *Single View Pose Estimation* lassen sich dabei in *Template-basierte*- (s. Abschn. 2.4.1), *Teileklassifaktions*- (s. Abschn. 2.4.2), und *Regressionsverfahren* (s. Abschn. 2.4.3) unterteilen.

Als Gegenstück zur *Single View Pose Estimation* hat sich im Bereich der *Full DOF Hand Pose Estimation* das *Model-based Pose Tracking* (s. Abschn. 2.5) etabliert. Dabei wird ein parametrisiertes Modell genutzt, um dessen bildliche Darstellung durch die kontinuierliche Transformation, Rendering und Fehlermessung an eine observierte Hand anzupassen. Ist die Übereinstimmung zwischen Modelldarstellung und Bild hoch, beschreiben die Modellparameter die Pose der Hand eindeutig.

Als Basiskomponenten eines modellbasierten Verfahrens werden meist ein Strukturmodell (s. Abschn. 2.5.1), ein Kinematikmodell (s. Abschn. 2.5.2), und ein Observationsmodell (s. Abschn. 2.5.3) genutzt. Das Strukturmodell dient dabei zur Beschreibung der Form der Hand und wird während der Erkennung über die Modellparameter angepasst. Im Kinematikmodell (s. Abschn. 2.5.2) werden die Anzahl und Art der Modellparameter festgelegt und welche Transformationen des Strukturmodells diese bewirken. Zudem besteht im Kinematikmodell die Möglichkeit den Wertebereich der einzelnen Modellparameter über *Constraints* zu beschränken und so eine einfache Dimensionsreduzierung zu erreichen (s. Abschn. 2.5.2). Als Kinematikmodell eignen sich neben denen, die das Bewegungsspektrum des realen Handskelletes (s. Abschn. 2.5.2) nachempfinden, *Disjoint-Modelle* (s. Abschn. 2.5.2). Hierbei kann die Bewegung einzelner Handteile unabhängig voneinander erfolgen, was ein parallelisiertes Tracking ermöglicht. Allerdings erfordert ein *Disjoint-Modell* eine explizite Modellierung der Zusammenhänge zwischen einzelnen Teilen durch Constraints.
Im Observationsmodell (s. Abschn. 2.5.3) wird schließlich festgelegt wie der Fehler zwischen einer Modelldarstellung gemessen werden soll.

Mit Hilfe der drei Basismodelle lässt sich nun eine Parametersuche formulieren, die als Optimierung des von den Modellparametern abhängigen Fehlers angesehen werden kann. Um den Aufwand dabei möglichst minimal zu halten und die Erkennung generell robuster gegenüber Mehrdeutigkeiten und Bildstörungen zu machen bedarf es eines optimalen Ansatzpunktes für die Suche. Im Sinne einer temporalen Kontinuität innerhalb der Erkennungssequenz liefert das jeweils letzte Ergebnis der Posenerkennung solch einen Ansatzpunkt für die Suche im aktuellen Aufnahmeschritt. So lässt sich die Suche meist durch ein Tracking beschreiben (s. Abschn. 2.5.4).
Als sehr robust gelten Verfahren, die ein *Multiple Hypothese Tracking* (s. Abschn. 2.5.4) realisieren, wie bspw. der Partikelfilter. Dabei werden mehrere Parameterkonstellation als Hypothesen über die Erkennungssequenz propagiert. Erweist sich dabei eine Hypothese als nicht korrekt, stehen noch andere zur Verfügung. Da die einzelnen Hypothesen unabhängig voneinander sind, müssen hier allerdings meist sehr viele Hypothesen gebildet, ausgewertet und und propagiert werden, was zu Laufzeitproblemen führen kann.

Da ein Tracking immer als rekursiver Filter formuliert ist, bedarf es in Situationen, in denen keine Historie von Ergebnissen zur Verfügung steht, einer Initialisierung (s. Abschn. 2.5.5). Dazu eignen sich im praktischen Einsatz insbesondere Verfahren aus dem Bereich der *Single View Pose Estimation*.

2.6.1 Abgrenzung

In dieser Arbeit soll eine möglichst allgemeingültige *Full DOF Hand Pose Estimation* durch ein *Model-based Pose Tracking* realisiert werden. Das Strukturmodell wird dabei durch ein *Polygon Mesh* definiert, da dies eine detailliertere Beschreibung der Handform zulässt.
Die Transformation, die Modelldarstellung und der Vergleich mit einer Observation soll dabei dediziert in *Shader-Programmen* auf einer handelsüblichen GPU erfolgen.
Als Kinematikmodell soll ein *Skeleton Model* mit 27-DOF-Parametern zum Einsatz kommen. Die Transformation des Strukturmodells soll ähnlich wie bei (Bray u. a., 2004) durch ein lineares *Skinning* (s. Abschn. 3.3.6) erfolgen.
Als Aufnahmegerät dient hier der Kinect-Sensor von Microsoft, der eine eine parallele Verarbeitung von Farb-und Tiefenbildern ermöglicht. Aus den Aufnahmebildern soll eine explizite tiefenbasierte Segmentierung der Handregion erfolgen, wobei dies durch ein einfaches Hautfarbenmodell unterstützt wird. Die Fehlermessung erfolgt in dieser Arbeit –wenn möglich– durch einen direkten pixelbasierten Vergleich zwischen der, im Tiefenbild segmentierten, Handregion und einem *gerenderten* Tiefenbild des Modells erfolgen. Für Fälle, in denen kein direkter pixelbasierter Vergleich möglich ist, kann eine Distanztransformation genutzt werden.
Inspiriert durch die Lösung von Oikonomidis u. a. (2011b) wird auch in dieser Arbeit die *Particle Swarm Optimization* zum Parameter-Tracking verwendet, da diese sich diese durch ihre einfache und intuitive Implementierung auszeichnet und zudem eine robuste Optimierungsstrategie verfolgt.
Zur Vereinfachung soll die Initialisierung des Tracking in dieser Arbeit nur rudimentär durch eine Initialisierungspose erfolgen. So soll die Hand mit ausgestreckten Fingern in eine vorab definierte Bildregion bewegt werden.

3 Design und Realisierung

Aufbauend auf der groben Zielvorgabe im Abschnitt 2.6.1 soll in diesem Kapitel die Realisierung der modellbasierten Handposenerkennung vorgestellt werden. Während der Realisierung wird hierbei zwischen einer *offline* Modellierungs- bzw. Vorbereitungsphase und der *online* Erkennungsphase unterschieden (s. Abb. 3.1).

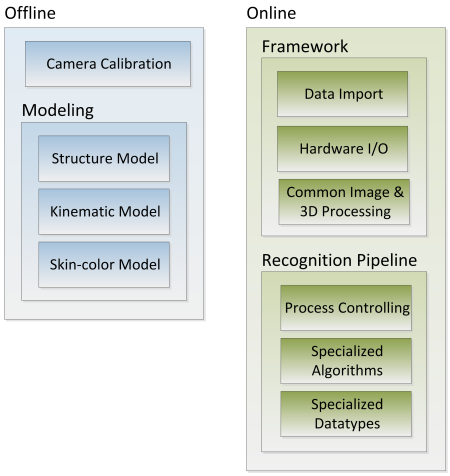


Abbildung 3.1: Systemarchitektur im Überblick

Die *offline* Phase (s. Abschnitt 3.2) umfasst vor allem die grafische Modellierung des Strukturmodells der Hand und die Beschreibung der Handdynamik durch ein Kinematikmodell (s. Abschn. 2.5.2). Um die Projektion der Hand in die Bildebene möglichst exakt über das Rendering des Handmodells nachempfinden zu können, müssen die Projektionsparameter des Aufnahmesystems bekannt sein. Diese werden ebenfalls in der *offline* Phase durch einen zusätzlichen Schritt der Kamerakalibrierung ermittelt (s. Abschn 3.2.1).
Zudem erfolgt in der *offline* Phase die Definition eines statistischen Hautfarbenmodells (s. Abschn. 3.2.3), das ebenfalls während der Posenerkennung zum Einsatz kommt.

Der *online* Teil (s. Abschnitt 3.3) der Entwicklung bezeichnet die eigentliche Implementierung der Handposenerkennung. Die Handposenerkennung basiert dabei auf dem *Particle Swarm Optimization Algorithmus* s. Abschn. 3.3.9). Die zur Erkennung notwendigen Schritte (s. Abschn. 3.3.2) sind dabei durch ihre sequenzielle Ausführung gekennzeichnet, sodass man hier von einer mehrstufigen Erkennungs-Pipeline sprechen kann. Die einzelnen Stufen der Pipeline

werden in dieser Arbeit durch das gängige Pipes-and-Filter-Pattern beschrieben, wobei jede Stufe als Komponente eines Frameworks (s. Abschn. 3.3.3) beschrieben wird. Die in der *offline* Phase erzeugten Modelldaten und der Zugriff auf die Grafikkarte und das Aufnahmegerät werden ebenfalls durch entsprechende Komponenten im Framework gekapselt.
In den Abschnitten 3.3.4 bis 3.3.9 des *online* Teils erfolgt schließlich eine ausführliche Beschreibung der Erkennungsfunktionalität.

Für den Betrieb der Handposenerkennung sind jedoch einige Voraussetzungen zu erfüllen, die sich sowohl an die Plattform als auch an den Einsatz dedizierter Hardware richten. Im nächsten Abschnitt 3.1 soll daher zunächst auf diese eingegangen werden.

3.1 Systemvoraussetzungen und Hardwareauswahl

Diese Arbeit sieht eine Entwicklung und den Einsatz einer 32-Bit-Bibliothek unter MS Windows 7 vor. Die Implementierung der Bibliothek erfolgt in C/C++, wobei als Hardwareplattform die x86/x64-Architektur eines konventionellen *Quad-Core*-PCs dient. Eine zusätzliche Plattformunabhängigkeit soll im Rahmen dieser Arbeit nicht gewährleistet werden. Allerdings wird die Möglichkeit der einfachen Portierung der Anwendung im Design berücksichtigt.

Die kontinuierliche Transformation des Handmodells und die anschließende Messung des Anpassungsfehlers zwischen Modelldarstellung und Observation erfolgt in dieser Arbeit durch angepasste *Shader*-Programme auf einer handelsüblichen Grafikkarte. Für die Entwicklung steht dabei eine DirectX11 taugliche AMD Radeon HD 6870[1] zur Verfügung.Als Aufnahmegerät wird in dieser Arbeit der multimodale Kinect-Sensor von Microsoft (Microsoft, 2013b) gewählt.
In dem folgenden Unterabschnitt soll auf die technischen Eigenschaften des Kinect-Sensors kurz eingegangen werden, da diese insbesondere für die Implementierungsphase relevant erscheinen.

3.1.1 Technische Eigenschaften der Microsoft Kinect und deren Softwareunterstützung

Die Microsoft Kinect (Microsoft, 2013b) kann als multimodaler Eingabesensor aufgefasst werden. Dabei sind für die in dieser Arbeit realisierte Handposenerkennung insbesondere die Farb- und Tiefenaufnahme der Kinect von Bedeutung. Weitere Ausstattungsmerkmale (s. Abb. 3.2) wie die Raummikrofone, der Schwenkmotor oder das

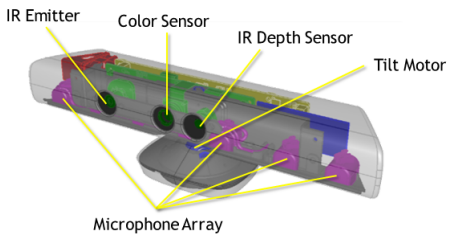


Abbildung 3.2: Ausstattung der Kinect (i.A.a. msdn.microsoft.com, 2013)

[1] http://www.amd.com/

interne Accelerometer[2] werden in dieser Arbeit nicht eingesetzt. Zur Aufnahme eines Farbbildes ist die Kinect mit einer konventionellen RGB-Kamera ausgestattet (s. Abb. 3.2). Das Tiefenbild der Kinect wird nicht direkt aufgenommen, sondern durch ein von Prime Sense patentiertes *Light-Coding*-Verfahren(Freedman u. a., 2010) aus dem Bild einer Infrarotkamera bestimmt.

Für die offizielle softwareseitige Unterstützung der Kinect bietet Microsoft das Kinect for Windows SDK (Microsoft, 2013c) an, das auch in dieser Arbeit verwendet wird.

Tiefenmessung durch Light Coding

Basis des *Light-Coding*-Verfahrens ist ein in die Umgebung projiziertes strukturiertes Infrarot-Punktmuster (*Speckle Pattern*), das gleichzeitig von der IR-Kamera aufgenommen wird (s. Abb. 3.3). Durch die Parallaxenunterschiede zwischen Projektor und Aufnahme und der definierten Struktur des Punktmusters lässt sich die Umgebungstiefe messen (Castaneda und Navab, 2011). Insgesamt kann daraus ein komplettes Tiefenbild der aufgenommenen Umgebung erstellt werden, wobei jeder Pixel des Tiefenbildes die Entfernung eines Punktes der Bildebene der IR-Kamera zum nächstliegenden Objektepunkt der Aufnahme darstellt.

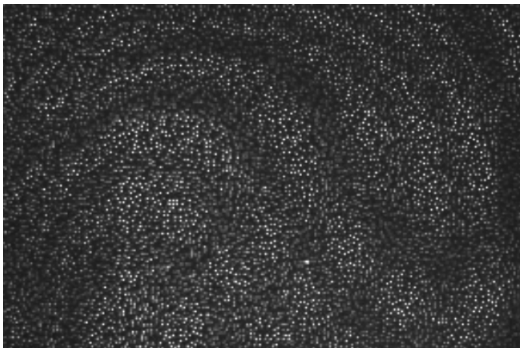


Abbildung 3.3: IR-Punktmuster der Kinect.

Das *Light-Coding*-Verfahren der Kinect erlaubt eine relativ hohe Bildwechselfrequenz der Tiefenaufnahme von 30 fps (*Frames per Second*). Microsoft (2013c) gibt dabei einen Entfernungsbereich vor, in der die Tiefe eines Objektes durch das *Light-Coding*-Verfahren genau bestimmt werden kann. Dieser liegt standardmäßig im Bereich von 800 mm bis 4000 mm (s. Abb. 3.4). Für die Nahaufnahme kann ein sog. *Near Mode* als Tiefenmessmodus eingestellt werden, bei dem der messbare Wertebereich der Tiefe auf 400 mm bis 3000 mm festlegt wird. In dieser Arbeit wird allerdings der standardmäßige Tiefenmessbereich gewählt, wobei ein Wert von 0 als ungültiger und 4095 mm (= 12 Bit) als maximaler Tiefenwert betrachtet wird.

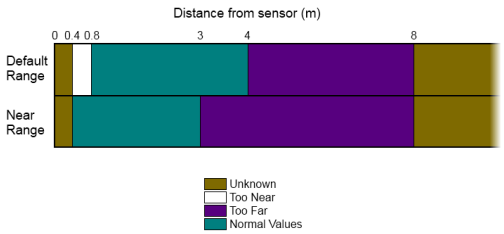


Abbildung 3.4: Tiefenmessbereich der Kinect. (i.A.a. Microsoft, 2013c).

[2] Beschleunigungssensor

Kinect for Windows SDK (Software Development Kit)

Die offiziellen Treiber der Kinect werden von Microsoft innerhalb des Kinect for Windows SDK (Microsoft, 2013c) geliefert[3]. Zur softwareseitigen Kommunikation mit dem Treiber enthält das *SDK* die die Programmierschnittstelle namens *NUI-API* (*Natural User Interface API*), die sowohl eine Unterstützung für C/C++ als auch für .Net anbietet.

Das SDK sieht eine Aufnahme und Bereitstellung der Infrarot- der daraus abgeleiteten Tiefen- und Frabbildern durch kontinuierliche Videoströme vor. So werden die von der Kinect aufgenommenen Bilder nebenläufig vom Treiber in einem Puffer gespeichert. Die gespeicherten Bilder können anschließend durch entsprechende Funktionen der NUI-API ausgelesen werden[4] (s. Abb. 3.5).

(a) Farbbild

(b) Infrarot-Bild.

(c) Tiefenbild (Tiefenpixel als Grauwerte interpretiert).

Abbildung 3.5: Kinect Aufnahmen.

Bei der Verarbeitung der Farb- und Tiefen-Bilder besteht ein Problem darin, dass die Linse der RGB- und der IR-Kamera unterschiedliche Kenndaten (Camera Intinsics) in Bezug auf die Brennweiten und Projektionszentren und Verzeichnung aufweisen. Zudem ist die RGB-Kamera neben der IR-Kamera angeordnet, wodurch die IR- und Farbaufnahme aus unterschiedlichen Blickwinkeln erfolgt (Parallaxenfehler). Um beide Aufnahmen in einheitlichen Pixelkoordinaten betrachten zu können, bedarf es daher einer Transformation (*Remapping*) der einzelnen Pixel des Farb-/Tiefenbildes in den Tiefen-/Farbbildbereich, was

Abbildung 3.6: Unterschiedliche Projektion des Farb- und Tiefenbildes.

[3] aktuell in Version 1.7

[4] Hier sei anzumerken, dass sich die Aufnahme der Farbkamera und die der Infrarotkamera einen Videostrom teilen. So lassen sich IR-Bilder und Farbbilder nicht gemeinsam aufnehmen.

eine Kenntnis der Projektionsparameter der IR-Kamera und der Farb-Kamera erforderlich macht. Die NUI-API gibt hier zwar nominale Brennweiten vor und liefert auch allgemeine Funktionen zum Remapping, allerdings können die exakten Projektionsparameter von Kinect zu Kinect je nach Herstellungsart variieren. Da die vorgestellte Lösung genauer bestimmte *Camera Intrinsics* voraussetzt, nicht nur um die Koordinaten des Farbbildes und Tiefenbildes zu vereinheitlichen, sondern vielmehr um ein Modell der Hand möglichst realistisch in eine Bildebene projizieren zu können, wird in dieser Arbeit eine explizite Kalibrierung der Kameras vorgenommen (s. Abschnitt 3.2.1).

3.2 Modellierungs- und Vorbereitungsphase (*offline*)

In der *offline* Modellierungs- und Vorbereitungsphase werden die, für den Betrieb der Handposenerkennung notwendigen, Modelldaten des Struktur-, Kinematik- und des Hautfarbenmodells erstellt und aufbereitet. Zudem werden hier die Projektionsparameter der Tiefen- und Farbkamera der Kinect über den Schritt einer Kamerakalibrierung ermittelt.

Zu Beginn der Vorbereitungsphase steht die Kalibrierung der Kameras an. Diese muss vor der Festlegung des Struktur- und Kinematikmodells erfolgen, da zu deren Modellierung 3D-Modell-Vorlagen, die sich aus den einzelnen Projektionsparametern und ausgewählten Tiefenaufnahmen der Hand ableiten, genutzt werden.

Während der grafischen Modellierung der Handstruktur werden die, genutzt, um die Modellform möglichst realistisch an die reale Form der Hand anzunähern. Zudem kann der Bewegungsablauf einzelner Teile (z.B. Finger) über die, aus mehreren Tiefenbildern der Hand erstellten, Vorlagen nachvollzogen werden und zur näherungsweisen Formulierung von kinematischen Constraints genutzt werden.

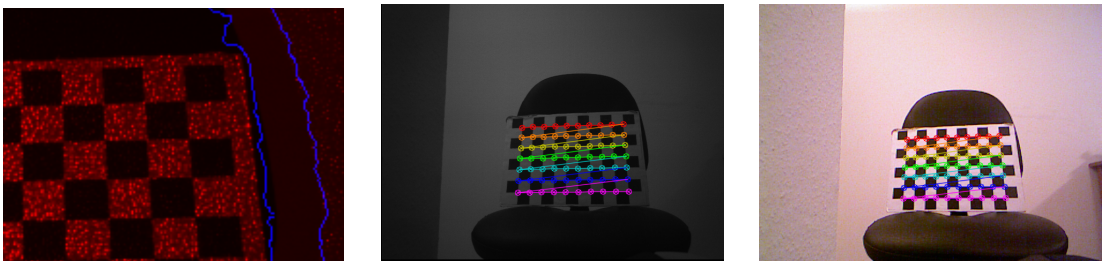
Des Weiteren erfordert die Posenerkennung die Formulierung eines einfachen statistischen Hautfarbenmodells. Dafür wird in dieser Arbeit das von Jones und Rehg (1999) beschriebene *Gaussian Mixture Model* der Hautfarbe verwendet.

3.2.1 Kalibrierung

Zur Kalibrierung der Kameras werden in dieser Arbeit die angebotenen Funktionen des Moduls zur Kamerakalibrierung und 3D-Rekonstruktion[5] der quelloffenen *OpenCV*-Bibliothek (opencv.org, 2013) genutzt. Die Kalibrierung der Kamera erfolgt hier durch die von Bradski und Kaehler (2008) vorgeschlagene Kamerakalibrierungs-Methode. Diese nutzt im Wesentlichen die, leicht analytisch zu beschreibende, Projektion von Punkten, die sich auf einer im physischen Raum befindlichen Ebene verteilen, um die Parameter (Intrinsics) eines idealisierten

[5] Dokumentation unter docs.opencv.org

Lochkameramodells herzuleiten. Die Berechnung der Intrinsics erfordert dabei, dass die einzelnen relativen physischen Koordinaten der Punkte und andererseits deren Projektion in die Bildebene der Kamera in Form von Pixelkoordinaten als Lösungs- und Wertemenge des Kameramodells bekannt sind. Um diese Punkte auf der Ebene und die Ebene selbst zu definieren, wird häufig eine an ein Schachbrett angelehnte Vorlage (Burros, 2013; Petersen, 2013) genutzt, wobei die einzelnen Eckpunkte der Schachfelder die zu betrachtenden Koordinaten darstellen.



(a) Horizontaler Versatz zwischen IR- und Tiefenbild. Die innere Kante (blaue Linie links) des Tiefenbildes stimmt nicht mit der Kante des Schachbretts des IR-Bildes (roter Hintergrund) überein.

(b) Schachbrett-Kalibrierung des IR-Bildes mit erkannten Eckpunkten. Die Extraktion der Eckpunkte erfordert das ausschalten des IR-Projektors.

(c) Schachbrett-Kalibrierung des Farbbildes mit erkannten Eckpunkten.

Abbildung 3.7: Kalibrierung von Tiefen-/IR- und Farbkamera.

In dieser Arbeit wird zur Kalibrierung der Farb- und der Tiefenkamera ebenfalls ein, aus unterschiedlichen Perspektiven aufgenommenes, Schachbrett genutzt (s. Abb. 3.7). Da diese Methode sich nur schwer auf die Tiefenwahrnehmung beziehen lässt, erfolgt die Kalibrierung der (virtuelle) Tiefenkamera anhand des Bildes der IR-Kamera. Das IR-Bild weist nur eine kleine konstante horizontale Verschiebung im Vergleich zum Tiefenbild (s. Abb. 3.7a) auf. Was während der Aufnahme und Kalibrierung durch eine einfache Translation des Bildes korrigieren lässt.
Die aus der Kalibrierung der einzelnen Kameras gewonnen Parameter reichen bereits aus, um 3D-Koordinaten jeweils auf die Pixel der Farbbild- und der Tiefenbildebene zu projizieren und umgekehrt. Um jedoch die Tiefenpixel auf die gleichen Koordinaten des Farbbildes abzubilden, bedarf es eines Ausgleichs des Parallaxenfehlers (s. Abschnitt 3.1.1) zwischen den beiden Kameras. Dies kann durch eine Transformation der physischen Koordinaten in das Sichtfeld der jeweiligen Kamera erreicht werden. Zur Bestimmung dieser Transformation zwischen dem Sichtfeld der Tiefenkamera und dem des Farbbildes wird, wie bei Burros (2013), die Stereokalibrierungs-Funktion der OpenCV-Bibliothek genutzt.

Das Ergebnis der Kalibrierung wird in Abb. 3.8 ersichtlich. Das Remapping des Tiefenbildes liefert im nahen Bereich gute Übereinstimmungen mit dem Farbbild (s. Schachbrett in Abb. 3.8). Im fernen Bereich nimmt die Genauigkeit der Übereinstimmung jedoch ab (s. Stuhl in Abb. 3.8).

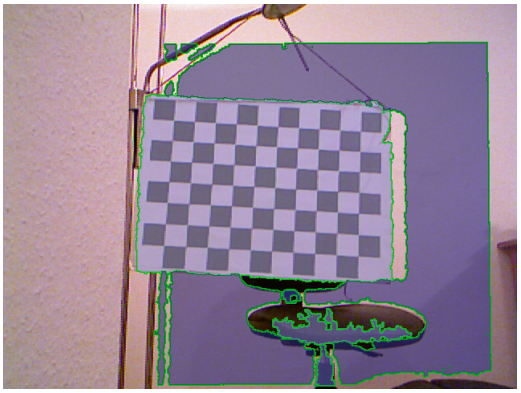


Abbildung 3.8: Ergebnis der Kalibrierung. Durch die Bestimmung der Kamera-Intrinsics und der Transformation zwischen den beiden Objekträumen der Kameras, lässt sich das Tiefenbild (hier blaue Überlagerung mit entfernten ungültigen Werten und grün hervorgehobenen Objektkanten) auf das Farbbild (hier Hintergrund) projizieren (Remapping).

3.2.2 Erstellung von Struktur- und Kinematikmodell der Hand

Nach der Kalibrierung der Kameras kann die Modellierung der Hand erfolgen. Zur Modellierung der Handstruktur kommt in dieser Arbeit Autodesk 3DS Max zum Einsatz (Autodesk, 2013a). Da bei der Erstellung der Handstruktur eine möglichst reale Handform nachempfunden werden soll, werden vor der eigentlichen Modellierung der Hand, aus ausgewählten Tiefenbildern einer Hand innerhalb der 3DS Max Umgebung, einzelne 3D-Modellierungsvorlagen durch ein selbst entwickeltes Script erstellt.
In Autodesk 3DS Max erfolgt dann auch die Definition des skelettbasierten Kinematikmodells, das, wie bei Bray u. a. (2004), durch ein *lineares Skinning* eine Transformation des Strukturmodells zulässt.
Der Export der Modelldaten erfolgt im standardisierten COLLADA-Format (Khronos Group, 2013).

Tools und Bibliotheken in der *offline* Phase

Autodesk 3DS Max

3DS Max[6] (Autodesk, 2013a) ist ein bekanntes etabliertes Animationsprogramm bzw. Autorenwerkzeug für 3D-Content. Zur Modellierung der Hand-Struktur und -Kinematik wird in dieser Arbeit 3DS Max 2011 verwendet. Dabei werden insbesondere die bereitgestellten Werkzeuge zur Erstellung, Manipulation und Transformation von einzelnen Vertices und Polygo-

[6] früher auch 3D Studio Max

nen eines Polygon-Mesh eingesetzt. Zur Modellierung der Handkinematik werden weiterhin die Funktionen von 3DS Max im Umgang mit dynamischen *Bones-Systems* genutzt (Autodesk, 2013b). Zur Beschreibung der Zusammenhänge zwischen statischer Handstruktur und der bone-basierten Kinematik wird dann der sog. *Skin-Modifier* benötigt. Zur Erstellung von 3D-Modellierungsvorlagen wird zudem die 3DS Max interne Scripting Umgebung MAXScrip genutzt (Autodesk, 2013c). Der finale Export der fertigen Modellierungsdaten in ein COLLADA-Format (Khronos Group, 2013) erfolgt ebenfalls durch Funktionen, die bereits in der 3DS Max-Umgebung integriert sind.

COLLADA und COLLADA Document Object Model

COLLADA ist ein XML-basiertes offenes Datenformat, das sich auf den einfachen Austausch von 3D-Inhalten zwischen Autorenwerkzeug und verarbeitenden Programmen spezialisiert. Standardisiert wurde das Format u.a. von der Khronos Group, die auch als Herausgeber fungiert (Khronos Group, 2013). Innerhalb des COLLADA-Standards erlaubt die *COLLADA Intermediate Language* eine Kodierung nahezu sämtlichen 3D-Content, der durch ein Autorentool erstellt oder durch ein Konsumentenprogramm verwendet werden kann. Der durch die *COLLADA Intermediate Language* kodierte 3D-Content kann dabei durch das *COLLADA Digital Asset and FX Exchange Schema* als Entität eines XML-Dokumentes beschrieben und validiert werden.

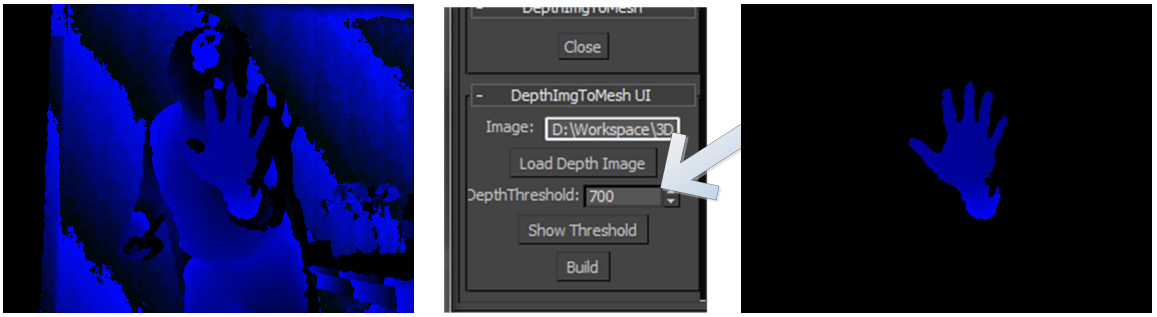
In dieser Arbeit erfolgt ein Austausch der Handmodelldaten zwischen 3DS Max und dem Erkennungssystem über eine einzelne COLLADA-XML-Datei (Modelldatei). Die Speicherung der Modelldatei erfolgt dabei automatisch durch entsprechende Export-Funktionen innerhalb von 3DS Max. Zum Import der Modelldatei im Erkennungssystem wird dagegen die quelloffene *COLLADA Document Object Model*-Bibliothek (*collada-dom*) genutzt (collada dom.sourceforge.net, 2013), die ein Auslesen[7] der einzelnen COLLADA-kodierten XML-Elemente nach dem bekannten, vom *W3C* standardisierten *Document Object Model DOM* ermöglicht.

Erstellung von 3D-Modellierungsvorlagen

Zur Erstellung der 3D-Modellierungsvorlagen wird in dieser Arbeit ein Script innerhalb von 3DS Max implementiert, das aus ausgewählten vorab aufgenommenen Tiefenbildern der Hand ein Polygonmodell erzeugt. Dieses kann innerhalb der 3DS Max Umgebung als eigenständiges Objekt betrachtet, transformiert und angepasst werden.

Das Script lädt zunächst ein bereits gespeichertes Tiefenbild und führt ein einfaches *Depth Thresholding* aus, um die Hand aus dem Bild zu segmentieren (s. Abb. 3.9). Der Wert des *Threshold* wird dabei manuell angegeben (s. Abb. 3.9b).
Ist die Hand segmentiert, lassen sich durch die, aus der Kamerakalibrierung (s. Abschnitt 3.2.1) ermittelten, inneren Projektionsparameter der Tiefenkamera und die gespeicherten Tiefeninformation des einzelnen Pixel des Tiefenbildes die zweidimensionalen bildlichen Koordinaten der segmentierten Handregion einfach in ihre physischen dreidimensionalen Ursprungskoordinaten

[7] Generierung und Anpassung von COLLADA-XML funktioniert natürlich auch.



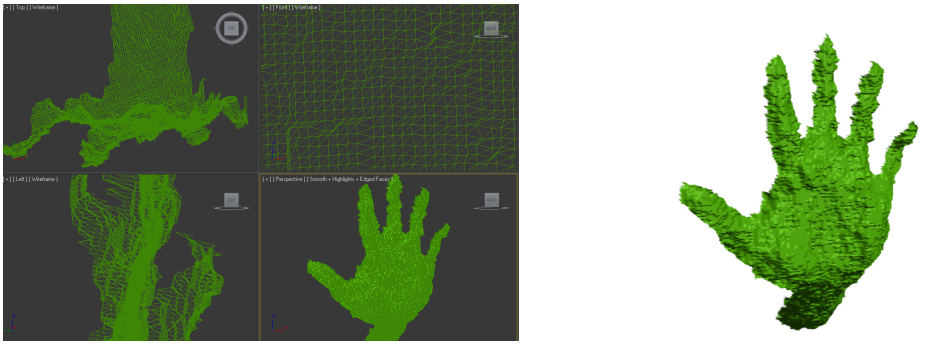
(a) Ausgewähltes Tiefenbild der Hand (Tiefenpixel als BGR-Pixel interpretiert.)

(b) Einstellung des *Depth-Threshold*.

(c) Ergebnis der Segmentierung.

Abbildung 3.9: MAXScript-basierte Segmentierung der Hand im Tiefenbild

zurück transformieren. Die Rücktransformation der Tiefenpixel der Handregion ergibt dabei eine Punktwolke, wobei jeder Punkt auf der realen Oberfläche[8] der aufgenommenen Hand liegt. Die Punkte der Wolke werden dann unter Berücksichtigung ihrer gerasterten Ausgangslage im Tiefenbild durch ein einfaches Verfahren trianguliert. Das Ergebnis der Triangulation ist ein Polygon-Mesh, das sich aus der Menge der Vertices, in Form der Punktwolke, und der, durch die Triangulation bestimmten, Faces zusammensetzt.



(a) Vorlagen Mesh in Oben- (links-oben), Vorder- (rechts-oben), Seiten- (links-unten) und Perspektivenansicht (rechts-unten). Vorlagen Mesh weist eine hohe Anzahl von Vertices und Faces (Dreiecke) auf.

(b) Oberfläche des Mesh ist stark verrauscht.

Abbildung 3.10: Resultierendes Mesh der Tiefenbild-Triangulation

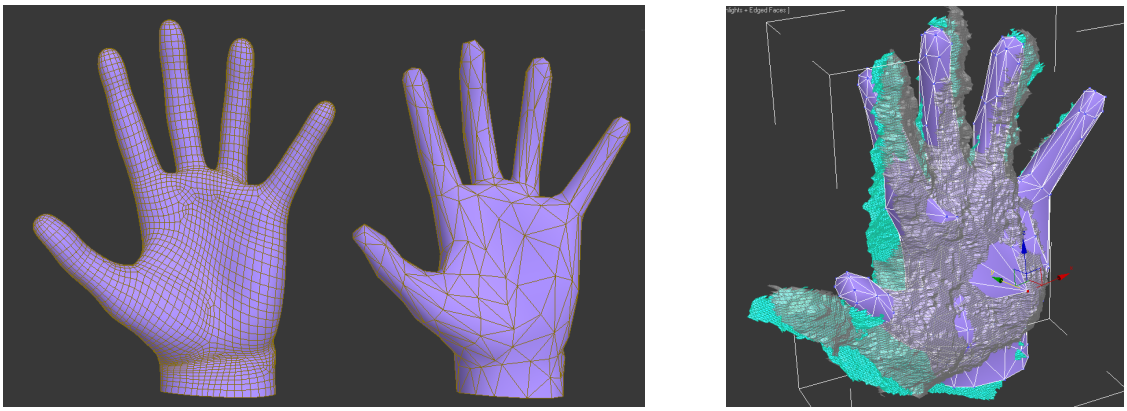
Da zur Generierung dieses Mesh die komplette ungefilterte Pixelmenge der Handregion einbezogen wird, setzt sich die Oberfläche des Mesh aus einer hohen Anzahl von Dreiecken zusammen (s. Abb. 3.10a). Weiterhin ist die Oberfläche des Mesh stark verrauscht (s. Abb. 3.10), was auf die nicht perfekte Messung und Auflösung der Tiefe durch die Kinect zurückzuführen

[8] nicht komplettes 3D-Volumen sondern nur aktuelle Perspektive der Tiefenaufnahme

ist. Diese Defizite der Mesh-Oberfläche und die Tatsache, dass das Mesh aus einem einzelnen Tiefenbild erstellt wird und daher kaum als allgemeine Repräsentation der Handoberfläche aus einem bestimmten Blickwinkel gelten kann, führten in dieser Arbeit zu der Entscheidung, dass das Mesh nicht direkt als Teil des Strukturmodells verwendet wird, sondern lediglich als Vorlage für eine manuelle Modellierung dient, in der sich die Handgeometrie allgemeiner und einfacher beschreiben lässt.

Strukturmodellierung

Da die Modellierung aus dem Nichts heraus eine Kunst für sich ist, wurde im Rahmen dieser Arbeit nach einem frei erhältlichen 3D-Modell gesucht, das bereits eine grobe Struktur der Hand vorgibt und sich zur manuellen Anpassung an die erstellten Vorlagen eignet. Dieses wurde im ZBrush-Forum von Pixologic (ZBrushCentral: *Free Hand OBJ*. www.zbrushcentral.com (2013)) gefunden. Da das Modell eine sehr hohe Polygonanzahl aufweist, was eine manuelle Bearbeitung sehr aufwändig gestalten würde, wird die Anzahl durch entsprechende Optimierungsfunktionen innerhalb von 3DS Max zunächst auf eine überschaubare Größe reduziert (s. Abb. 3.11a).



(a) High-Polygon Modell als Basis des Strukturmodells (Quelle: www.zbrushcentral.com). Zur einfacheren manuellen Aufbereitung der Geometriedaten wird die Polygonanzahl des Ursprungsmodells (links) stark reduziert (rechts).

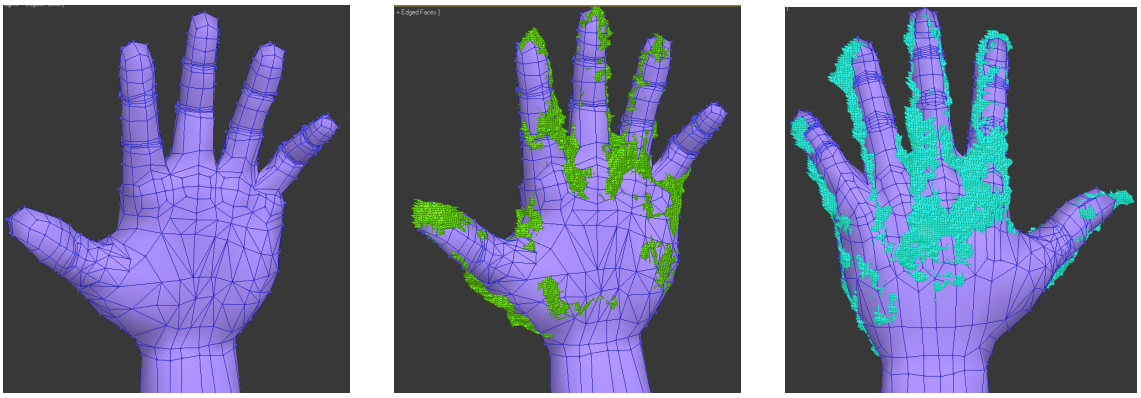
(b) Anpassung des Handmodells an einzelne Vorlagen. Vertices des Handmodells (violett) werden auf die Vorlagen der Rückansicht (türkis) und der Vorderansicht (transparent) verschoben

Abbildung 3.11: Anpassung eines Handmodells

Die Aufgabe der Strukturmodellierung beschränkt sich so auf die Anpassung der vorgegebenen Geometrie des entliehenen Modells an die einzelnen Modellierungsvorlagen. Dies geschieht im Wesentlichen dadurch, dass an detaillierteren Stellen der Hand neue Vertices mit entsprechenden Polygonen eingefügt, das Polygon-Netz umstrukturiert und die einzelnen Vertices des Basismodells annähernd auf die Oberfläche der erstellten Vorlagen verschoben werden. Da die Oberflächen der einzelnen Vorlagen durch die nicht optimale Tiefenmessung der Kinect einige

Defizite und keine hundertprozentige Übereinstimmung der Handproportionen aufweisen, gilt für die Anpassung des Handmodells, dass die unterschiedlich vorgegebenen Handproportionen zu einem gemeinsamen Gesamteindruck zusammengefasst und offensichtliche Fehler der einzelnen Vorlagenoberfläche ausgeglichen werden müssen (s. Abb. 3.11b).

Das Ergebnis der Strukturmodellierung kann in Abb. 3.12 betrachtet werden. Der Unterschied zwischen den einzelnen Modellvorlagen und dem daraus abgeleiteten approximiertem Strukturmodell lässt zudem eine erste grobe Beurteilung des während der Posenerkennung zu erwartenden Fehlers zu (s. Abb. 3.12b und 3.12c). Insgesamt weist das resultierende Strukturmodell 3489 Vertices und 2329 Faces auf. Die Anzahl der Geometriedaten lässt sich bei Bedarf weiter reduzieren.



(a) Strukturmodell der Hand. (b) Fehler des Strukturmodells zur Vorlage der Vorderansicht. (c) Fehler des Strukturmodells zur Vorlage der Rückansicht.

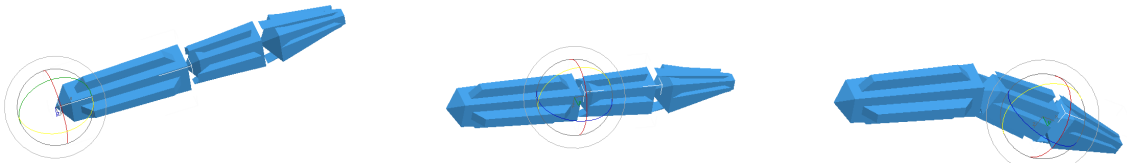
Abbildung 3.12: Ergebnis der Strukturmodellierung

Modellierung der Handkinematik

Als Basis des Kinematikmodells dient in dieser Arbeit wie bei Bray u. a. (2004) ein virtuelles Handskelett, das über ein lineares Skinning eine Transformation des Strukturmodells zulässt.

Bones System zur Beschreibung des Handskeletts:

Die Nachbildung des Handskeletts und deren Bewegungsgrade (DOF) erfolgt innerhalb der 3DS Max Umgebung durch ein *Bones System* (Autodesk, 2013b). Die einzelnen Bones des Systems lassen sich dabei in gleicher Weise verketten, wie es die Knochenstruktur der realen Hand vorgibt. Durch diese Verkettung weist das Bones System ebenfalls eine Hierarchie auf, in der sich die Bewegung eines einzelnen Bones immer auch auf dessen Nachfolger innerhalb der Hierarchie auswirkt (s. Abb. 3.13).

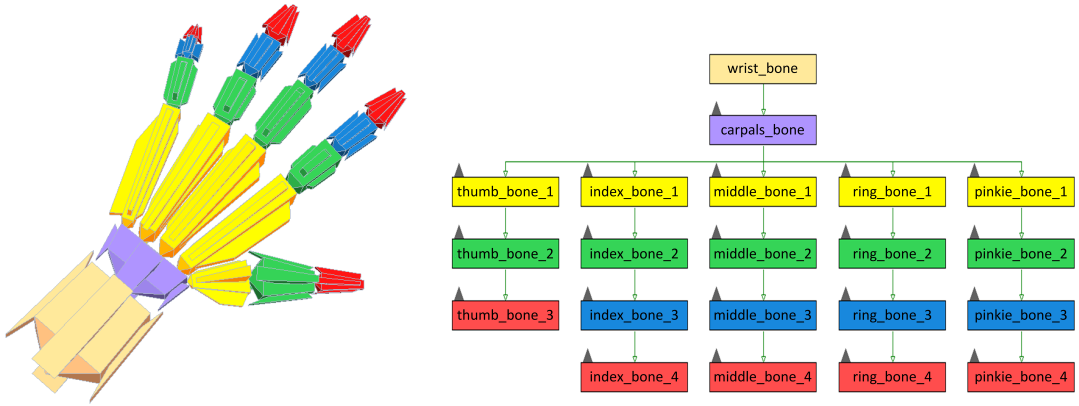


(a) (b) (c)

Abbildung 3.13: Bone Hierarchie. Die Bewegung eines einzelnen Knochens führt zur Bewegung der Nachfolger.

Zur Definition des Handskeletts werden die einzelnen Bones unter Berücksichtigung ihrer Abhängigkeiten erstellt und ihre Längen an die aus dem Strukturmodell ersichtlichen Handproportionen angepasst (s. Abb. 3.14). Um das kinematische Modell dabei möglichst allgemein und realistisch zu halten, wird hier jeder Knochen (inklusive Karpalknochen und einem rudimentären Handgelenksknochen) des realen Handskeletts durch ein entsprechendes Bone-Objekt abgebildet. Insgesamt ergeben sich dadurch 21 *Bones* für das Kinematikmodell (s. Abb. 3.15) mit einer in Abb. 3.15b ersichtlichen Hierarchie.

Abbildung 3.14: Bones Systems (blau) auf Grundlage des Strukturmodells (grau transparent).



(a) Bone System reflektiert die reale Knochenstruktur der Hand.

(b) Resultierende Bone Hierarchie. Einzelne Bones sind a) farblich zugeordnet.

Abbildung 3.15: Resultierendes Handskelett beschrieben durch ein Bones System

Strukturtransformation durch Skinning:

Um die Geometrie des bisher statischen Strukturmodells entsprechend des bewegungsbeschreibenden Handskellets transformieren zu können, muss eine Zuordnung zwischen den einzelnen Geometriedaten und Bones des Handskeletts erfolgen. Dies geschieht in der Regel durch ein lineares Skinning (Bray u. a., 2004; Autodesk, 2013b). Hierbei wird jedem Vertex des Strukturmodells ein Gewichtungsfaktor für jeden Bone des Handskeletts zugeordnet. Ein Gewichtungsfaktor gibt dabei an, wie stark ein Vertex von einem entsprechenden Bone abhängt und wie weit dessen Position durch die Bewegung des Bone verändert wird. Die Summe aller Gewichte eines Vertex ist auf eins normiert, womit sichergestellt ist, das ein Vertex mindestens einem Bone zugeordnet ist.

Innerhalb der 3DS Max-Umgebung wird ein Skinning durch Anwendung des *Skin Modifier* (Autodesk, 2013b) auf das erstellte Polygon-Mesh der Hand erreicht. Durch den Skin Modifier wird dabei bereits eine grobe Verteilung der Gewichte anhand der relativen Positionen der Vertices zu den Bones vorgegeben. Für optimale Ergebnisse müssen die Gewichte anschließend jedoch noch einmal manuell angepasst werden (s. Abb. 3.16). Das Ergebnis des Skinning ist ein parametrisiertes Strukturmodell, das sich über die lokale Rotation der einzelnen Bones, ähnlich wie die reale Hand, verformen lässt (s. Abb. 3.17).

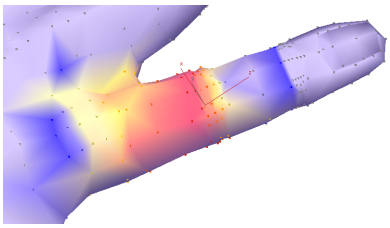
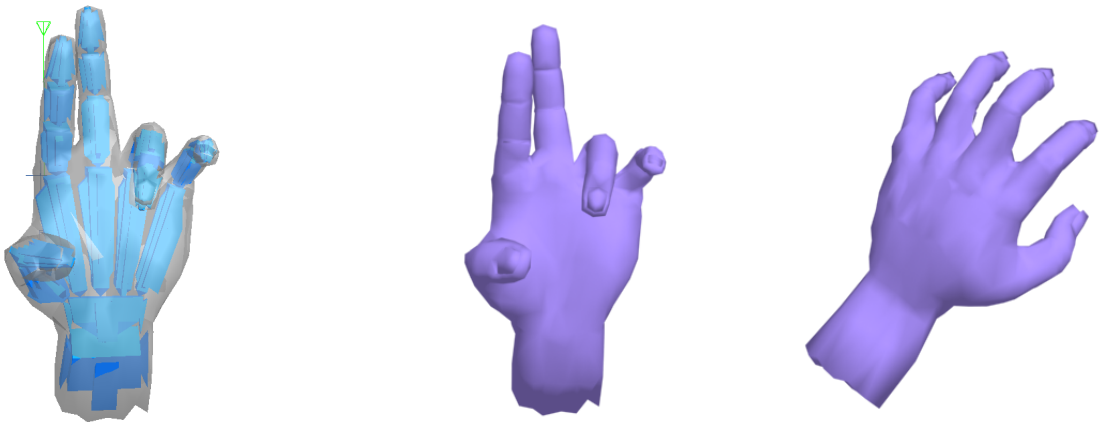


Abbildung 3.16: Skinning Weights für den Bone des unteren Zeigefingerglied. Absteigender Einfluss des Bones von rot-orange bis blau-grau.



(a) Verformung des Strukturmodells (transparent grau) durch Bewegung der Bones (blau).

(b) Strukturmodell gemäß a) ohne Darstellung von Bones.

(c) Weitere Strukturmodell-Konfiguration.

Abbildung 3.17: Ergebnis des Hand-Skinning

Parametrisierung

Das bisher beschriebene Handmodell lässt eine
uneingeschränkte Transformation der einzelnen Bones und damit der Geometrie der Hand zu. Dies ergibt eine hohe Anzahl von Modell-DOF und kann zudem viele ungewollte Modellkonfigurationen liefern, die keiner realen Grundlage entspringen. So werden im *online* Teil von vornherein nur einzelne Transformationen bei der Verwendung des Modells zugelassen. Diese Transformationen lassen sich so als die einzelnen Modellparameter bzw. DOF auffassen.

Die Auswahl der erlaubten Transformationen orientiert sich hier grundsätzlich an der von Albrecht u. a. (2003) beschriebenen Lösung zur realen Animation der Hand. So kommen bei der Verwendung des Modells im *online* Teil dieser Arbeit nur lokale Rotationen zur Transformation der einzelnen Bones zum Einsatz.

Die Bones der Fingerspitzen, der mittleren Fingerglieder, der Daumenspitze und des mittleren Daumenglieds dürfen dabei jeweils nur um die eigene lokale X-Achse rotiert werden, so dass jeder dieser Bones nur 1 DOF aufweist. Bei den unteren Fingergliedern wird zusätzlich die Rotation[9] um die lokale Y-Achse erlaubt, womit für die Bewegung jedes Fingers 4 DOF genutzt werden. Da die Metacarpale (Knochen in der Handfläche) der Finger nur eingeschränkt beweglich sind, werden diese zur Vereinfachung während der Modelltransformation ignoriert. Das bedeutet, dass diese Bones nur von der Transformation des jeweils übergeordneten Bones in der Hierarchie beeinflusst werden, sich selbst allerdings nicht durch Parameter anpassen lassen. Dies gilt allerdings nicht für den Metacarpalknochen des Daumens, der 3 DOF in Form der dreidimensionalen lokalen Rotation erhält. Der Carpalknochen (Knochen nach Handgelenksknochen) dient in dieser Arbeit zur Beschreibung der globalen Lage der Hand. Dazu werden 3 lokale Rotations-DOF genutzt. Der Wrist-Bone dient hier lediglich zur dreidimensionalen, durch 3 DOF ausgedrückten, globalen Translation der Hand.
Insgesamt ergibt sich damit eine DOF- bzw. Parameteranzahl des Modells von 27 (s. Abb. 3.18). Für den *online* Teil gilt dabei, die beschriebene Transformationen der einzelnen DOF möglichst performant umzusetzen.

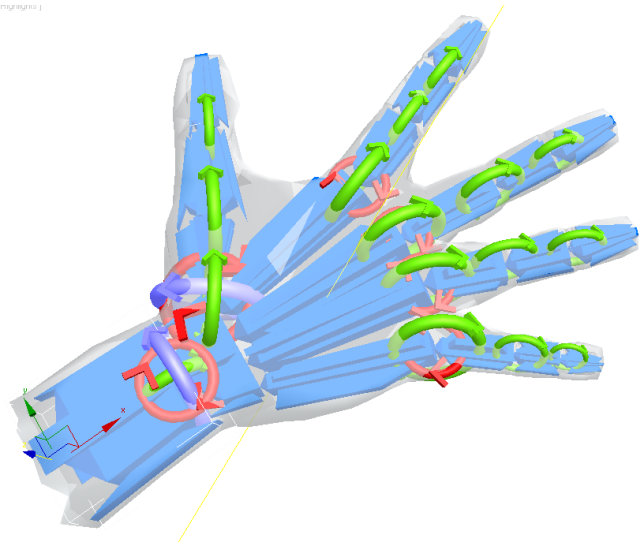


Abbildung 3.18: DOF des Handmodells.

[9] Albrecht u. a. (2003) sehen hier drei DOF vor.

Constraints

Der Einfachheit halber werden in dieser Arbeit einfache statische *Domain Constraints* für jeden einzelnen Parameter formuliert. Diese beschränken die Rotationswinkel eines Bone-Parameters lediglich auf den Bereich zwischen zwei konstanten Grenzen, ohne dass die gesamte Hierarchie der Bones betrachtet wird. Dies führt dazu, dass während des Tracking in der *online* Phase auch Parameterkonstellationen in Betracht gezogen werden können, die keiner realen Grundlage entsprechen, und daher das Risiko für Fehldetektionen erhöhen und die Effizienz des Tracking negativ beeinflussen können. Daher sollte in späteren Versionen eine exaktere Constraint-Modellierung vorgesehen werden.
Die Constraints in Form der Parameter-Grenzen werden in dieser Arbeit empirisch bzw. visuell innerhalb der 3DS Max-Umgebung bestimmt.

Bereitstellung und Verwendung des Handmodells

Das, durch das Strukturmodell und das Kinematikmodell, beschriebene Handmodell wird durch die COLLADA Export Funktion von 3Ds Max in einer entsprechend formatierten XML-Datei lokal gespeichert. Die wesentlichen Daten, die innerhalb der COLLADA-Datei beschrieben werden sind in Abb. 3.19 angegeben.

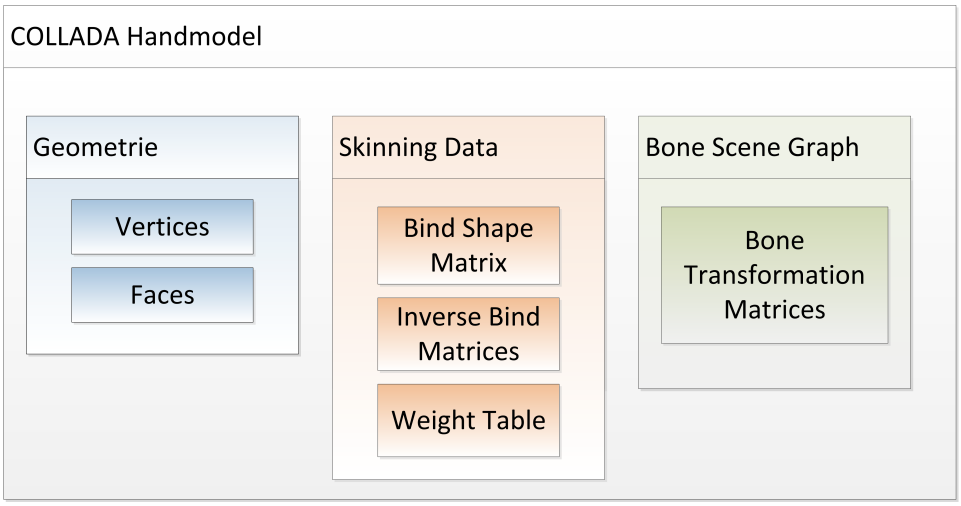


Abbildung 3.19: Wesentliche Daten des COLLADA Handmodells.

Neben der, durch das Strukturmodell vorgegeben, reinen Geometrie wird im COLLADA-Format noch die Boneshierarchie des Kinematikmodells durch einen gerichteten Szenengraph und die Skinningparameter beschrieben (Barnes und Finch, 2008). Der Bones-Szenegraph speichert dabei für jeden Bone eine relative Transformationsmatrix RBT_i, die deren Lage und Position relativ zu seinem Vorgänger-Bone[10] angibt. Die gesamte Menge der angegebenen Transforma-

[10] Für den ersten Bone des Graphen –der Wrist-Bone– bezieht sich die Transformationsmatrix auf das physische Koordinatensystem (World)

tionsmatrizen beschreibt so eine Skelettkonfiguration, die als initiale Handpose oder *Bind Pose* bezeichnet wird.

Um ein Skinnining der Geometrie in der *online* Phase durchführen zu können, beschreibt das COLLADA Format neben den einzelnen Skinning-Gewichten (s. Abschn 3.2.2) noch weitere Skinningparameter. Dazu zählt einerseits die Bone-unabhängige *Bind Shape Matrix* (BSM), die die Transformation der Geometrie des Strukturmodells in den Koordinatenraum des definierten Handskeletts beschreibt. Wesentlicher ist hier jedoch die Angabe der *Inverse Bind Matrices* (IBM_i). Diese werden jedem einzelnen Bone zugeordnet und dienen zur absoluten Rücktransformation eines bereits bewegten Bones in seine ursprüngliche, innerhalb der Bind Pose beschriebenen, Ausgangslage. So besteht immer die Möglichkeit die aktuelle Lage und Position eines Bones zurückzusetzen und durch Verwendung einer aktualisierten Transformationsmatrix neu zu positionieren.

Um die Modellinformationen der *online* Handposenerkennung zur Verfügung zu stellen, wird für den *online* Teil dieser Arbeit ein spezieller *Model-Importer* entworfen, der die angebotene Funktionalität der COLLADA-DOM-Bibliothek (collada dom.sourceforge.net, 2013) zum Analysieren und Auslesen der Daten des COLLADA-Modells nutzt. Die so extrahierten Daten werden dabei durch entsprechende Komponenten innerhalb des Erkennungs-Framework repräsentiert (s. Abschn 3.3.3).
Da die Transformationen der einzelnen Bones in Form der lokalen Rotationen um ihre Verbindungsstellen dynamisch zur Laufzeit erfolgen soll, lassen sich diese nicht vorab innerhalb des COLLADA-Modells exportieren[11]. So erfolgt die Transformation der einzelnen Bones online durch ein explizit in dem Erkennungsframework beschriebenes Kinematikmodell (s. Abschnitt 3.3.5).

3.2.3 Erstellung des Hautfarbenmodells

Um eine Messung des Fehlers zwischen Modelldarstellung und Aufnahmebild nur auf die, durch die Tiefenpixel beschriebenen, Handoberfläche zu beziehen soll in dieser Arbeit vor dem eigentliche Erkennungsschritt die Handregion aus dem gelieferten Tiefenbild segmentiert werden. Dies erfordert eine vorangehende Filterung des Eingabefarbbildes durch ein statisches Hautfarbenmodell. Das Ergebnis kann dann zur eigentlichen Segmentierung der Handregion im Tiefenbild verwendet werden.
Das verwendete Hautfarbenmodell basiert auf den Erkenntnissen von Jones und Rehg (1999). Jones und Rehg (1999) erstellen dabei aus einem Satz von 18.696 aufbereiteten Fotos ein *Skin-Color Model* und ein komplementäres *Non-Skin-Color Model*. Die Modelle werden in der Arbeit durch zwei, über den RGB-Farbraum definierte, Histogramme repräsentiert und können so zur statistischen Einordnung einzelner RGB-Farbpixel in eine Hautfarben-Klasse genutzt werden. Die beiden Histogramme wurden dabei approximiert als *Gaussian Mixture Models* angegeben so dass auf diese zurück gegriffen wird.

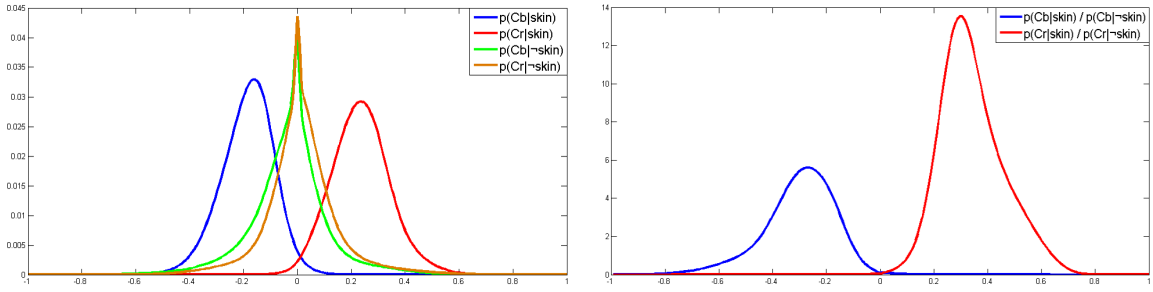
[11] Im gegensatz zu Keyframe Animation

Durch die *Mixture Models* lassen sich dabei bedingte Wahrscheinlichkeitsverteilungen der Hautfarbenklasse $p(rgb|skin)$ und der Nicht-Hautfarbenklasse $p(rgb|\overline{skin})$ analytisch, in approximierter Weise beschreiben. Um während der *online* Erkennung nicht kontinuierlich die Funktion der *Mixture Models* für jeden Pixel eines Aufnahmebildes ausführen zu müssen, werden die bedingten Wahrscheinlichkeiten als Ergebnisse des *Gaussian Mixture Models* für jeden möglichen RGB-Wert vorab berechnet und in einer *Lookup Table*[12] gespeichert. Um die Größe der *Lookup Table* LUT von vornherein zu reduzieren und um das Hautfarbenmodell unabhängig von Helligkeitsunterschieden (Luminanz) zu beschreiben, erfolgt während der Berechnung der beiden *Lookup Tables* eine Umbettung der RGB-Werte in den CbCr-Raum des YCbCr-Farbmodells (s. Gl. 3.1). durch eine Konvertierungsfunktion ($rgbToYCbCr$). Die einzelnen Einträge der LUT, die jeweils durch eine CbCR-Konstellation indiziert werden, lassen sich dann unter Berücksichtigung der *Gaussian Mixture Model*-Funktion sequenziell durch eine Akkumulation der bedingten Wahrscheinlichkeiten berechnen (s. Gl. 3.2).

$$CbCr = rgbToYCbCr(RGB) \tag{3.1}$$

$$LUT_i^{CbCr} = LUT_{i-1}^{CbCr} + \sum_{i=1}^{N} w_i * \frac{1}{(2\pi)^{\frac{3}{2}}|\Sigma_i|^{\frac{1}{2}}} \mathrm{e}^{-\frac{1}{2}(RGB-\mu_i)^T \Sigma_i^{-1}(RGB-\mu_i)} \tag{3.2}$$

Die Parameter für jeden Model-*Kernel* i in Gl. 3.2 in Form des skalaren Gewichtungsfaktors w_i, der einzelnen Einträgen der Kovarianz Matrix Σ_i und des *Mean-Vektor* μ_i werden für das Skin-Model und das Non-Skin-Model im Anhang von Jones und Rehg (1999) aufgeführt.
Die so berechnete LUT repräsentiert schließlich die in Abb. 3.20 angegebene Wahrscheinlichkeitsverteilung.



(a) Wahrscheinlichkeitsverteilungen separat für den Cb- (blau für Hautfarbe, grün für Nicht-Hautfarbe) und den Cr-Kanal (rot für Hautfarbe, orange für Nicht-Hautfarbe) der YCbCr-Farbwerte.

(b) Wahrscheinlichkeitsquotient (Likelihood-ratio) zwischen Hautfarben- und Nicht-Hautfarben-Klasse für den Cb-(blau) und dem Cr-Kanal (rot). Kann zum Klassifizieren von Pixeln verwendet werden.

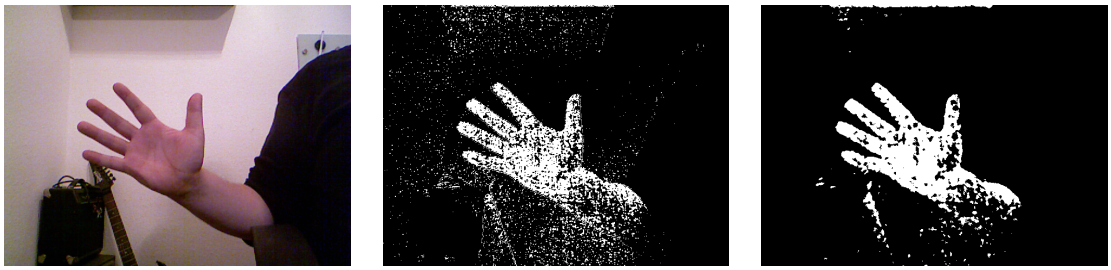
Abbildung 3.20: Wahrscheinlichkeitsverteilung der Haut- und Nicht-Haufarben-Klasse im CbCR-Farbraum.

[12] Kein Histogramm, da die bedingte Wahrscheinlichkeit ja bereits analytisch beschrieben ist.

Aus den *Lookup Tables* lassen sich während der Handposenerkennung für einen in das YCBCR-Format umgewandelten RGB-Pixel die bedingten Wahrscheinlichkeiten $p(rgb|skin)$ und $p(rgb|\overline{skin})$ auslesen und durch einen einfachen *Likelihood-ratio Test* (Jones und Rehg, 1999; Phung u. a., 2005) auf Zugehörigkeit zur Hautfarbenklasse überprüfen (s. Gl. 3.3).

$$\frac{p(rgb|skin)}{p(rgb|\overline{skin})} \geq \tau \tag{3.3}$$

Der Threshold τ in Gl. 3.3 wird dabei in dieser Arbeit empirisch ermittelt.
Ein Test des Hautfarbenmodells liefert bei einem Threshold von 1.5 die in Abb. 3.21 aufgezeigten Ergebnisse.



(a) Kinect-Aufnahme der Hand.

(b) Hautfarbentest ohne vorangehende Filterung. Das Ursprungsbild der verwendeten Kinect ist stark verrauscht und rotstichig. Dieser Umstand könnte eine Vorfilterung der Aufnahme nötig machen.

(c) Hautfarbentest eines Gaußweichgezeichneten Eingabebildes. Weniger Bildrauschen führt zu besseren Ergebnissen.

Abbildung 3.21: Test des CbCr-Hautfarbenmodells für einen Threshold τ von 1.5

Die Hautfarbenfilterung durch das resultierende Modell liefert aufgrund der nicht perfekten Farbaufnahme der Kinect[13] nur eine annähernd genaue Falsch-Positiv-Einordnung der einzelnen Farbpixel (s. Abb. 3.21), sodass die Filterung noch nicht als vollständige Segmentierung gelten kann. Die gefilterten Pixel des Modells werden daher in dieser Arbeit nur genutzt um die hypothetische Position einer Handregion im Eingabebild zu ermitteln, wobei das erstellte Hautfarbenmodell der Handposenerkennung als *offline* Modell zur Verfügung gestellt wird. Die eigentliche Segmentierung erfolgt explizit durch ein Tiefen-Thresholding innerhalb des Tiefenbildes (s. Abschn 3.3.4).

[13] Könnte durch eine explizite Farbkorrektur behandelt werden.

3.3 Online Erkennungsphase

Während der *online* Erkennungsphase soll die Handpose mithilfe der, *offline* Modelldaten, einer online Adaption des Kinematikmodells, eines online beschriebenen Observationsmodells und dem *Particle Swarm Optimization Algorithmus* kontinuierlich aus den Aufnahmebildern der Kinect erkannt werden. Dazu wird eine Erkennungs-Pipeline entworfen, die eine strom-basierte, sequenzielle Verarbeitung der Aufnahmebilder in einzelnen wohldefinierten Schritten realisiert.

Die Implementierung der Pipeline erfolgt gemäß dem *Pipes-and-Filter-Pattern* (Buschmann, 1996). Dabei sieht das *Pipes-and-Filter-Pattern* eine Abstraktion der, in den einzelnen Pipeline-Stufen beschriebenen, Verarbeitungsfunktionalität in Form von individuellen Filter-Komponenten vor (s. Abb. 3.22). Die allgemein gefasstere Schnittstelle zwischen den einzelnen Pipeline-Stufen in Form der Ausgabedaten der jeweiligen Stufe und weiteren Synchronisierungsfunktionen wird dagegen in Pipes-Komponenten gekapselt. Diese Trennung von Verarbeitungsfunktionalität und Schnittstellen hat den Vorteil, dass sich die, durch die Filter beschriebenen, Verarbeitungskomponenten mit relativ einfachen Mitteln austauschen lassen, ohne dass sich die generelle Struktur der Pipeline verändert.

Abbildung 3.22: Pipes-And-Filters-Pattern. Quelle: Wikipedia, de.wikipedia.org (2013)

Die einzelnd realisierten Pipes- und Filter, werden als Komponenten innerhalb eines Erkennungs-*Framework* gekapselt. Weiterhin werden in dem *Framework* übergreifend genutzte Datenstrukturen und Funktionen bereitgestellt sowie der allgemeine Zugriff auf die statischen offline Modelldaten und die Hardware beschrieben. Innerhalb des Framework erfolgt dann auch die Definition der, sowohl für das Tracking als auch zur Initialisierung der Handposenerkennung benötigten, online Modelle. Dazu zählt einerseits die Beschreibung eines dynamischen Skeletts innerhalb des Kinematikmodells und andererseits die Definition das Observationsmodell der Handposenerkennung.

3.3.1 Particle Swarm Optimization

Die *Particle Swarm Optimization* (*PSO*) stellt einen relativ neuen Algorithmus zur iterativen numerischen Optimierung einer nichtlinearen mehrdimensionalen Problemfunktion dar. Der Algorithmus, der ursprünglich zur Simulation des Sozialverhaltens schwarmorientierter Organismen (z.B. Fisch- oder Vogelschwärme) in Bezug auf deren Streben nach einem willkürlichen Zielort (z.B. Futterquelle) konzipiert wurde, lässt sich analog auf die allgemeine Suche eines Optimums

einer unbekannten Funktion übertragen. Die Übertragung und Anpassung der *Particle Swarm Optimization* auf ein allgemeineres Optimierungsproblem[14] wurde erstmals durch Kennedy und Eberhart (1995) bzw. Eberhart und Kennedy (1995) dokumentiert. Eine ausführliche Analyse und Abgrenzung des *PSO*-Algorithmus gegenüber anderen Optimierungsstrategien erfolgt in Wagner (2010). Eine Übertragung der *PSO* auf das konkrete Problem des modellbasierten Tracking von Handposen lässt sich dagegen in der Lösung von Oikonomidis u. a. (2011c) wiederfinden und kann so als Basis für die Entwicklung der Erkennungspipeline gelten.

Als Ausgangslage des *PSO*-Algorithmus dient eine Menge von Lösungen, die zu Beginn zufällig über den Lösungsraum verteilt werden, wobei die einzelnen Lösungen meist als Partikel bezeichnet werden. Auf die Handposenerkennung bezogen stellt der Lösungsraum (*Hyperraum*) (Eberhart und Kennedy, 1995)) den DOF- bzw. Modellparameterraum und die einzelnen Partikel jeweils eine konkrete DOF- bzw. Parameterkonstellation dar. Die einzelnen Partikel gelten dabei als zufällig erzeugte Hypothesen für einen einzelnen Tracking-Schritt. Für jeden der Partikel lässt sich, durch Auswertung der zu optimierenden Funktion, deren aktueller Zustand ermitteln (Fitnesswert). Zur Posenerkennung erfolgt dies durch die Messung des Fehlers innerhalb des Observationsmodells zwischen der, von einem Partikel, angegebenen Modellkonfiguration und der observierten Hand.
Durch den Algorithmus wird nun ein dynamisches Bewegungsmodell beschrieben, das die Partikel iterativ, durch deren schrittweisen Positionsveränderung gegen ein Optimum (minimaler Fehler) des Lösungsraumes konvergieren lässt. Die zur Konvergenz nötige Anpassung einer Partikelbewegung wird dabei über ein soziales Modell erreicht (Wagner, 2010). Dieses berechnet für jeden Partikel einen Geschwindigkeitsvektor, der sowohl die Richtung als auch die Schrittweite der Bewegungsveränderung berechnet. Zur Berechnung des Geschwindigkeitsvektors wird dabei die gesamte Menge der Partikel als einheitliches System betrachtet, sodass neben der Bewertung der einzelnen lokalen unkorrelierten Zustände der Partikel auch der globale Systemzustand, der sich aus der bisher besten gefunden Position aller Partikel ergibt, während der Anpassung der einzelnen Partikelpositionen berücksichtigt wird. So ergibt sich das Bewegungsmodell aus dem emergenten Verhalten des gesamten Partikelsystems, wobei dieses dem realen Vorbild eines biologischen Schwarms nachempfunden wird.

[14] Training eines Neuronalen Netzes

PSO-Algorithmus bezogen auf die Handposenerkennung

i. **Initialisierung:**
   1. *Erzeuge globale Variablen*:
      a. $\vec{pMin}_G$: Parameterkonstellation, die den minimalsten Fehler aller Partikel (global) lieferte.
      b. $eMin_G := +\infty$: Bisher gefundener minimalster Fehler aller Partikel (global).
   2. Erzeuge eine Menge von N zufällig im Modellparameterraum verteilter Partikel mit folgenden Eigenschaften:
      a. $\vec{p}_i := RNG$: Aktuelle Parameterkonstellation des Partikels i, zufällig initialisiert.
      b. e_i : Aktueller Fehler des Partikels i.
      c. $\vec{pMin}_i$: Parameterkonstellation, die den minimalsten Fehler für Partikel i lieferte.
      d. $eMin_i := +\infty$: Bisher gefundener minimalster Fehler des Partikels i.
      e. $\vec{v}_i := \vec{0}$: Aktueller Geschwindigkeitsvektor des Partikels i.

ii. **Fehleroptimierung für das Eingabebild I, Fehlerthreshold Θ und maximale Iterationsanzahl n:**
   1. *Führe folgende Schritte für jeden Partikel aus*:
      a. $e_i = f(I, \vec{p}_i)$: Berechnen des Fehlers für die aktuelle Parameterkonstellation des Partikels i durch Ausführung der Fehlerfunktion E.
      b. $(eMin_i, \vec{pMin}_i) = \begin{cases} (e_i, \vec{p}_i) & e_i < eMin_i \\ (eMin_i, \vec{pMin}_i) & e_i \geq eMin_i \end{cases}$: Aktualisiere den minimalen Fehler und die Parameterkonstellation des Partikels i.
      c. $(eMin_G, \vec{pMin}_G) = \begin{cases} (e_i, \vec{p}_i) & e_i < eMin_G \\ (eMin_G, \vec{pMin}_G) & e_i \geq eMin_G \end{cases}$: Aktualisiere den globalen minimalen Fehler und die Parameterkonstellation aller Partikel.
      d. $v_i = S(v_i)$: Aktualisiere den Geschwindigkeitsvektor des Partikels i durch Ausführung eines sozialen Modells S (s. Abschnitt 3.3.1).
      e. $\vec{p}_i = \vec{p}_i + \vec{v}_i$: Aktualisiere die Parameterkonstellation des Partikels i anhand des Geschwindigkeitsvektors.
      f. Falls $eMin_G < \Theta$ oder n erreicht, dann breche Optimierung ab und liefere $\vec{pMin}_G$ als Ergebnis. Ansonsten weiter mit Schritt ii..1.

Soziales Modell der PSO

Als Schwerpunkt der PSO kann die Berechnung des Geschwindigkeitsvektors durch ein soziales Modell angesehen werden. Der Geschwindigkeitsvektor wird dabei von dem sozialen Modell aus den unabhängig voneinander zu berechnenden Komponenten (Wagner, 2010), der Trägheit, der Kognition und der Sozialkomponente zusammengesetzt (s. Abb. 3.23).

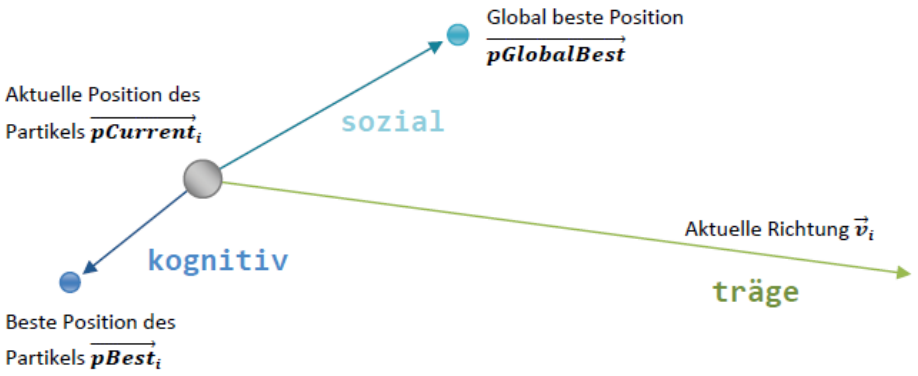


Abbildung 3.23: Geschwindigkeitsvektor als Resultat der Komponenten des sozialen Modells der PSO (i.A.a Wagner, 2010). Hier für das *GBEST*-Sozialmodell dargestellt.

Die Trägheitskomponente definiert dabei, inwieweit die aktuelle Bewegungsrichtung eines Partikels nach der Aktualisierung durch das soziale Modell beibehalten werden soll.
Die Kognitivkomponente gibt dagegen an, wie stark ein Partikel von seiner bisher lokal besten gefundenen Position angezogen werden soll, also inwieweit der Geschwindigkeitsvektor in Richtung der Parameterkonstellation $p\vec{M}in_i$ streben soll.
Die Sozialkomponente des Geschwindigkeitsvektors gibt an, inwieweit die Bewegungsrichtung eines einzelnen Partikels von den gemeinsamen Ergebnissen des Schwarms beeinflusst wird. Dadurch hat sie einen wesentlichen Einfluss auf das Konvergenzverhalten der PSO und damit auf die Effektivität der Optimierung.

Zur Berechnung des Geschwindigkeitsvektors stellen Eberhart und Kennedy (1995) zwei grundsätzliche soziale Modelle –das *GBEST*-Modell (*Global Best*) und das *LBEST*-Modell (*local Best*)– vor. Beim *GBEST*-Modell gibt die Sozialkomponente an, in welchem Maß der Partikel in Richtung der bisher besten globalen Position des gesamten Partikelschwarms streben soll. Also inwieweit sich die Parameterkonstellation eines Partikels der Konstellation $p\vec{M}in_G$ nähert. Das *LBEST*-Modell nutzt dagegen die Sozialkomponente, um die Bewegung eines Partikels an die Partikel in seiner direkten Nachbarschaft anzupassen. Allerdings kann bei beiden Modellen der Partikelschwarm stark auseinander driften (*Swarm Explosion*), was zu einem divergenten instabilen Verhalten des Algorithmus führt (Clerc und Kennedy, 2002). Clerc und Kennedy (2002) empfehlen daher eine Erweiterung des *GBEST*-Modells. Dabei dient eine Gewichtung durch einen speziellen *constriction factor* κ während der Berechnung des Geschwindigkeitsvektors dazu, den Bewegungsspielraum eines Partikels von vornherein einzugrenzen. Die Berechnung des Geschwindigkeitsvektors $\vec{v}_i^t$ eines Partikels erfolgt dann nach Clerc und Kennedy (2002) und Oikonomidis u. a. (2011c) rekursiv wie folgt:

$$\vec{v}_i^t = \kappa \cdot [\vec{v}_i^{t-1} + c_C \cdot (\vec{r}_1 \circ (p\vec{M}in_i - \vec{p}_i)) + c_S \cdot (\vec{r}_2 \circ (p\vec{M}in_G - \vec{p}_i))] \tag{3.4}$$

mit:

$$\varphi = c_C + c_S \tag{3.5}$$

$$\kappa = \begin{cases} \sqrt{\frac{2*\sigma}{\varphi - 2 - \sqrt{\varphi^2 - \varphi^4}}} & \text{für } \varphi > 4 \\ \sqrt{\sigma} & \text{sonst} \end{cases} \tag{3.6}$$

Die Terme $\vec{v}_i^{t-1}$, $c_C \cdot (\vec{r}_1 \circ (p\vec{M}in_i - \vec{p}_i))$, und $c_S \cdot (\vec{r}_2 \circ (p\vec{M}in_G - \vec{p}_i))$ in Gl. 3.4 stellen dabei jeweils den Geschwindigkeitsvektor des letzten Optimierungsschritts, der direkt als Trägheits-Komponente dient, die Kognitiv- und die Sozialkomponente dar.

$\vec{r}_1$ und $\vec{r}_2$ stellen Zufallsvektoren dar, die im Intervall zwischen $[0;1]$ liegen und über das Hadamard-Produkt[15] zu einer stochastischen Gewichtung der einzelnen Komponenten führen. So wird der der Partikelbewegung eine gewisse chaotische Eigenschaft zugeordnet, bei der je nach Zufall entschieden wird, welche Richtung ($p\vec{M}in_G$ oder $p\vec{M}in_i$) in einem Optimierungsschritt eingeschlagen werden soll.

c_C und c_S stellen zusätzliche skalare Gewichtungskonstanten für die Kognitiv- bzw. die Sozialkomponente dar. Diese sollten aus Gründen der Konvergenz möglichst so gewählt werden, dass $c_C + c_S > 4$ ist. Oikonomidis u. a. (2011c) nutzen hier $c_C = 2.8$ und $c_S = 1.3$.

κ stellt den eigentlichen *constriction factor* dar, der durch Gl. 3.6 vor Ausführung des Algorithmus berechnet werden kann. σ ist dabei einen Parameter im Intervall $]0;1]$, der laut (Clerc und Kennedy, 2002) zur weiteren Beeinflussung des Konvergenzverhaltens genutzt werden kann. So soll $\sigma = 1$ eine sehr intensive Suche nach einem Optimum ergeben, wobei hier mit einer langsameren Konvergenz zu rechnen sei. Da die PSO in dieser Arbeit in Form eines Tracking verwendet wird, und dadurch bereits eine Einschränkung des Lösungsraumes unter Verwendung der temporalen Kontinuität erfolgen kann, wird $\sigma = 1$ gewählt.

3.3.2 Die Erkennungs-Pipeline

Unter Verwendung der *Particle Swarm Optimization* wird folgende Pipeline umgesetzt (s. Abb. 3.24):

[15] Komponentenweise Vektor Multiplikation.

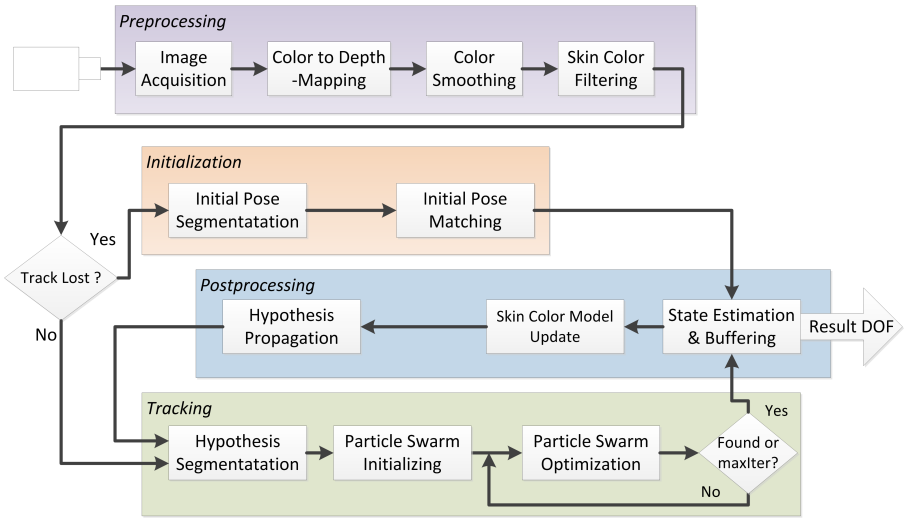


Abbildung 3.24: Erkennungs-Pipeline.

Innerhalb der Pipeline (s. Abb. 3.24) wird zunächst das aufgenommene Farb- und Tiefenbild der Kinect durch die Schritte einer einfachen Vorverarbeitung (*Preprocessing*) transformiert gefiltert und global gespeichert. So werden die Bilder zunächst durch den Schritt der *Image Acquisition* ausgelesen. Aufbauend darauf werden die Pixel des Farbbildes nacheinander durch die Schritte des *Color-to-Depth-Mapping* und des *Color Smoothing*, in entsprechender Reihenfolge in den Bildbereich des Tiefenbildes transformiert (s. Abschn. 3.2.1), und durch einen Gauß-Filter geglättet. Aus dem transformierten und geglätteten Farbbild werden dann innerhalb des *Skin-Color-Filtering*-Schrittes unter Verwendung des *Hautfarben-Modells* die hautfarbenen Pixel bestimmt (s. Abschn. 3.2.3). Die Filterung der Hautfarben-Pixel eignet sich jedoch je nach Zustand des *Hautfarben-Modells* nur bedingt zur Freistellung der kompletten Handregion, sodass eine exakte Segmentierung in separaten angepassten Schritten innerhalb der Initialisierungsphase und die Trackingphase der Pipeline erfolgt (s. Abschn. 3.3.4). Nach der Vorverarbeitung wird der globale Zustand der Pipeline überprüft (*Track-Lost?* s. Abb. 3.24) und entschieden, ob in die Initialisierungs- oder Tracking-Phase der Pipeline gewechselt werden soll.

Innerhalb der Initialisierungsphase erfolgt hier zunächst eine Segmentierung (*Initial Pose Segmentation*) der Handregion ausgehend von einer einer konstanten Position im Tiefenbild (s. Abschn. 3.3.4). Im darauf folgenden Schritt des *Initial Pose Matching* wird das Handmodell entsprechend einer initalen Handparameterkonstellation konfiguriert und deren Übereinstimmung zur segmentierten Handregion durch die einfache Ausführung des Observationsmodells (s. Abschn. 3.3.7) festgestellt.

Für die Trackingphase der Pipeline steht neben den Eingabebildern noch eine hypothetische Modellparameterkonstellation aus dem jeweils letzten Erkennungsschritt zur Verfügung. Dabei stellt die Hypothese im Sinne eines *Single Hypothesis Tracking* (Erol u. a., 2007) immer das unveränderte Ergebnis des letzten Pipeline-Durchlaufs dar. Die Hypothese wird jetzt dazu verwendet, um die Handregion innerhalb des Schrittes der *Hypothesis Segmentation* aus dem

Tiefenbild zu extrahieren (s. Abschn. 3.3.4) und in dem darauf folgenden Schritt der *Particle Swarm Initializing* zur Generierung der *PSO*-Partikel genutzt, wobei diese innerhalb eines festen Bereichs um den, durch die Hypothese gelieferten, Ausgangspunkt im Modellparameterraum verteilt werden (s. Abschn. 3.3.9).
Während der Verteilung der Partikel wird darauf geachtet, dass die, durch das Kinematik-Modell vorgegebenen, Constraints der Modellparameter erfüllt werden, sodass die Partikel zum großen Teil eine Parameterkonstellation repräsentieren, die einer realen Grundlage entspricht.

Im nachfolgenden Schritt der *Particle Swarm Optimization* erfolgt dann eine iterative Suche nach der passenden Modellparameterkonstellation durch die kontinuierliche Anpassung der Partikel gemäß des PSO- Algorithmus, wobei jede Modellparameterkonstellation eines Partikels zur Konfiguration des Handmodells verwendet wird und dieses anschließend mit der segmentierten Handregion über das Observationsmodell der Lösung verglichen wird. Die Optimierung erfolgt solange, bis ein entsprechendes Abbruchkriterium erfüllt ist (maximale Iteration ist erreicht oder der Unterschied zwischen Modell und Handregion ist kleiner als ein vorgegebener *Threshold*).

In der finalen *Postprocessing* Phase erfolgt schließlich die Ausgabe der Modellparameterkonstellation, die die Initialisierung bzw. das Tracking als Ergebnis liefern. Weiterhin erfolgt hier aufbauend auf den Erkennungsergebnissen innerhalb des *State Estimation & Buffering*-Schrittes eine Pufferung der Ergebnisse und die notwendige Einschätzung des Systemzustands gemäß der Frage: Ïst der Übereinstimmungsfehler zwischen Handmodell und Bildregion gering und wurde daher die Handpose erkannt?". Schließlich wird das Erkennungsergebnis für die Tracking-Phase des nächsten Pipeline-Durchgangs durch die *Hypothesis Propagation* als Hypothese bereitgestellt. Dabei werden in der vorgeschlagenen Lösung die gefundenen Modellparameter einfach weitergereicht. In aufbauenden Arbeiten könnte hier eine weitere Einschätzung der Folgeparameter durch einen dynamischen Filter (z.B. Kalman-Filter), der die jeweilige Handbewegung berücksichtigt, erfolgen.

3.3.3 Framework Architektur

Wie in Abb. 3.24 zu erkennen ist, lassen sich die einzelnen Schritte der Pipeline in die vier unterschiedlichen Phasen des *Preprocessing*, der Initialisierung, des Tracking und des *Postprocessing* unterteilen. Innerhalb des Erkennungsframework (s. Abb. 3.25) wird jede dieser Phasen durch eine eigenständige Pipeline über das *Pipes-And-Filter-Pattern* modelliert. Jeder Schritt der Pipeline entspricht dabei einer Filterkomponente, die die,in einer Pipe-Klasse gekapselten, Daten des vorangehenden Filters als Eingabe erhält und ihre Ausgabedaten in die entsprechend nächste Pipe schreibt.

Die Auswahl der Pipeline und die globale Überwachung des Erkennungsprozesses erfolgt in der *Pipeline Controller*-Klasse, hier wird also auch entschieden, ob eine Initialisierung oder das Tracking erfolgt. Der *Controller* dient zudem als globaler Speicher für sämtliche System-Objekte (Modelle, Pipelines, Filter und Pipes) und kann daher auch direkt auf die einzelnen Filter-Komponenten zugreifen.

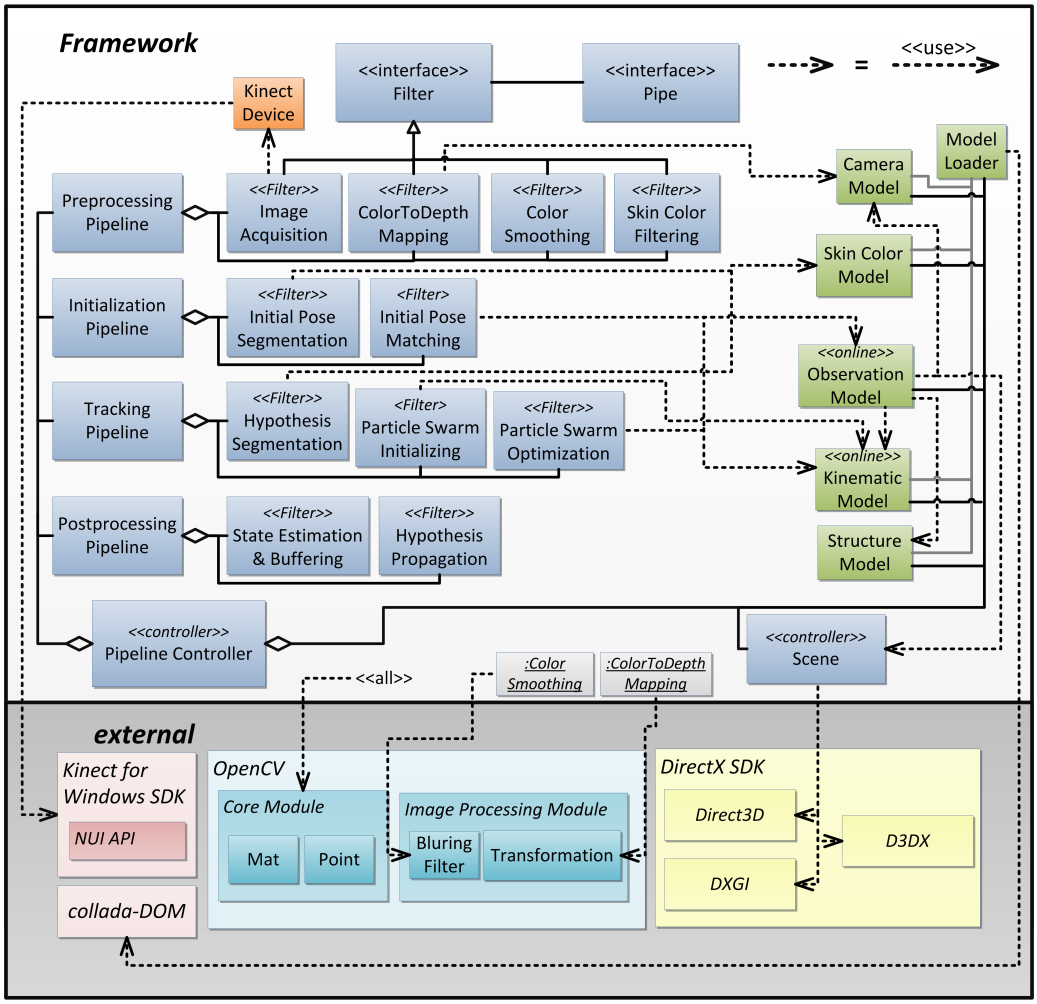


Abbildung 3.25: Framework Architektur.

Zum Laden der offline erstellten Modelldaten nutzt der *Pipeline Controller* eine *Model Loader Instanz*. Diese initialisiert einerseits die Instanz des *Camera-Models* mit den aus einer Textdatei geladenen Kalibrierungsdaten (s. Abschn. 3.2.1). Andererseits werden hier die Geometriedaten und Skinning-Informationen (s. Abschn. 3.2.2) unter Verwendung der *collada-DOM*-Bibliothek (s. Abschn. 3.2.2) aus der Collada-Datei geladen und zur Initialisierung des Struktur- und Kinematikmodells der Hand verwendet.

Die online Adaption des Kinematikmodells und das Observationsmodell werden einer kontinuierlichen Veränderung unterzogen, so dass diese als online Modelle gelten können, deren Funktionalität meist über die einer reinen Speichereinheit hinausgeht. Auf die Funktionalität der online Modelle und deren jeweiligen Abhängigkeiten wird dabei in den nachfolgenden Abschnit-

ten ausführlicher eingegangen, sodass hier auf diese verwiesen werden soll (s. Abschn. 3.3.5 und Abschn. 3.3.7).

Die Abhängigkeiten zwischen den einzelnen Filtern der Pipeline und den geladenen Modellen sind in Abb. (s. Abb. 3.25) ersichtlich. Für die Messung des Anpassungsfehlers zwischen Modell und Aufnahme bedarf es meist nur einer Ausführung einer in der *Observation Model*-Klasse definierten Funktion. Um die Modellparameter der einzelnen, innerhalb der *Particle Swarm Initializing* erzeugten, Partikel während der Initialisierung und der *Particle Swarm Optimization* auf die vorgegebenen Parameterberreiche zu begrenzen, wird diesen Filtern der direkte Zugriff auf die im Kinematik-Modell hinterlegten statischen Constraints ermöglicht. Eine Sonderrolle nimmt der *Image Acquisition*-Filter ein. Dieser nutzt eine *Kinect Device*-Klasse, die als Fassade zur Steuerung und zum Auslesen des Eingabebildstroms der Kinect dient. Innerhalb des Kinect Device wird hierbei im Wesentlichen die Funktionalität der NUI-API des Kinect for Windows SDK gekapselt (s. Abschn. 3.1.1).

Zur allgemeinen Abstraktion des Zugriffs auf die Grafikhardware wird ein zusätzlicher Controller in Form der *Scene*-Klasse bereitgestellt. Da Filter sowie Modelle gleichermaßen die Möglichkeit der Hardwarebeschleunigung durch die GPU erhalten sollen, muss damit gerechnet werden, dass die Daten und Shader-Programme innerhalb des Grafikspeichers häufiger ausgetauscht werden müssen. Dieser Wechsel des *Render-Kontexts* wird dabei durch die *Scene-Klasse* vereinfacht.

Externe Bibliotheken und Module

Neben den Bibliotheken der NUI-API des Kinect for Windows SDK (s. Abschn. 3.1.1) und der *collada-dom*-Bibliothek (s. Abschn. 3.2.2), auf die bereits eingegangen wurde, wird das Core-Modul und das Image Processing Modul der OpenCV-Bibliothek (opencv.org, 2013) sowie die D3DX- und Direct3D-API des DirectX SDK (Microsoft, 2013a) zur Realisierung des Frameworks benötigt.

OpenCV Klassen und Funktionen:
Die OpenCV-Bibliothek (opencv.org, 2013) stellt grundlegende Datentypen und einige Basisfunktionen im *Core-Modul* der Bibliothek zur Verfügung. In dieser Arbeit wird insbesondere die universell einsetzbare *Mat*-Klasse aus dem *Core-Modul* zur Repräsentation von bildlichen Daten verwendet. Neben *Mat* werden noch die *Point*- sowie die Vektor-Strukturen (*Vec*) in allen verfügbaren Dimensionen zur einheitlichen Repräsentation von diskreten und reellen 2D- 3D- und homogenen 4D-Koordinaten verwendet.

Neben den allgemeinen Funktionen zur Verarbeitung der bildlichen Mat-Klasse (Kopieren, Pixel setzen, binäre Operationen usw.) wird im Wesentlichen auf die Operationen des *Gauß*-Glättungsfilters und die Funktionen zur geometrischen Bildtransformation des *Image Processing*-Moduls der OpenCV-Bibliothek innerhalb des *Color-Smoothing-Filters* und des *Color-To-Depth-Mapping-Filters* des Frameworks zurückgegriffen. Die geometrisch affinen Bildtransfor-

mationen mit bilinearer Interpolation dienen dabei zum Remapping des Farbbildes anhand des Kamera-Modells.

DirectX SDK:
Zur Realisierung der Handposenerkennung wird das aktuell erhältliche DirectX SDK (Microsoft, 2013a) von Microsoft für Windows 7 verwendet. Innerhalb des Frameworks wird die vom SDK gelieferte *Direct3D-API*, die *D3DX-API*, sowie rudimentär das *DXGI*-Framework[16] verwendet.

Die Funktionen zur Steuerung der programmierbaren Verarbeitungs-Pipeline[17] einer DirectX-fähigen Grafikkarte werden in der *Direct3D-API* (*D3D*) beschrieben. So lassen sich mit der API die einzelnen Stufen der Grafik-Pipeline konfigurieren, Modelldaten (Vertices und Indices), bildliche Texturdaten, Konstanten und Parameter (z.B. Transformationsmatrizen) sowie Shader-Programme in den Grafik-Speicher transferieren und die Verarbeitungsdurchgänge starten. Innerhalb des Frameworks wird das Laden der Daten und der Shader-Programme und das Starten eines GPU-Verarbeitungsdurchgangs durch die *Scene*-Klasse vereinfacht.

Die *D3DX-API*[18] kann als mathematische Bibliothek betrachtet werden. Hierin findet man Strukturen, wie *Matrizen*, *Vektoren* und *Quaternionen* mit entsprechenden Operatoren. In dieser Arbeit werden die o.g. Strukturen hauptsächlich zur Transformation des Handskelletes innerhalb des *Kinematikmodells* verwendet.

Neben den C++-Biliotheken definiert das DirectX SDK mit der *HLSL* (High Level Shading Language) einen Sprachstandard zur Shader-Programmierung. Die für die Erkennung notwendigen Shader-Programme werden entsprechend in HLSL implementiert.

3.3.4 Segmentierung

Bevor eine Observation mit dem Handmodell verglichen werden kann, bedarf es in dieser Lösung einer Segmentierung der Handregion aus dem aufgenommenen Tiefenbild. Da das verwendete Hautfarbenmodell in Kombination mit der Farbaufnahme der Kinect zu einer nicht perfekten Segmentierung führt (s. Abschn. 3.2.3), wird hier eine Tiefensegmentierung auf Basis einer Hautfarbenvorfilterung vorgeschlagen. Als Ausgangslage für die Segmentierung dienen die im vorangehenden *Skin-Color-Filtering*-Schritt durch Anwendung des statischen Hautfarbenmodells, ermittelte Menge an hautfarbenen Pixel.

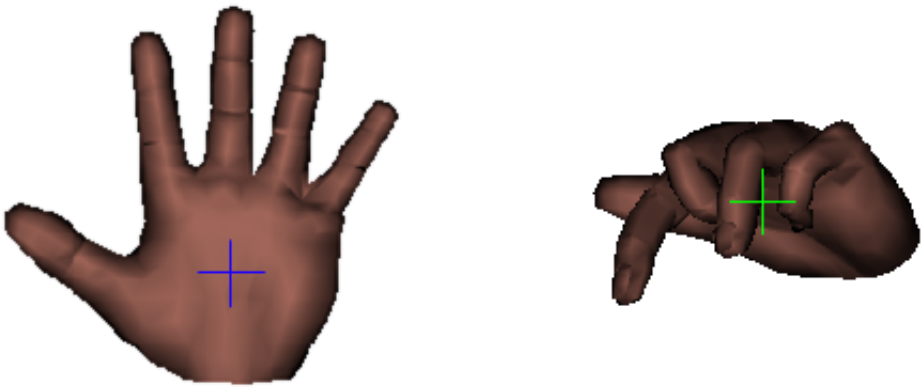
Zur Segmentierung wird aus der Menge der Haufarbenpixel ein Kandidat gewählt, dessen Bildkoordinaten (p_0) als *Seed Point* (Initialpunkt) für einen einfachen, über das Tiefenbild definierten, *Floodfill*-Algorithmus (Tönnies, 2005) genutzt werden. Die Auswahl dieses *Seed Point*

[16] Wird zum Erzeugen einer ID3D11Device- und ID3D11DeviceContext-Instanz benötigt.
[17] Darstellung der D3D11-Pipeline unter http://msdn.microsoft.com/en-us/library/ff476882(2013)
[18] Die *D3DX-API* gilt unter Windows 8 als *deprecated* und wird durch die *DirectXMath* Bibliothek ersetzt. Der Einfachheit halber kommt in dieser Arbeit noch die *D3DX-API* zum Einsatz.

kann als lokale Suche nach einem geeigneten Haufarbenpixel in der Umgebung eines angenommenen Hypothesenpunktes p_h, der sich idealerweise immer in der Nähe der observierten Handregion befindet, aufgefasst werden. Der Hypothesenpunkt wird in der Initialisierungs- und Tracking-Phase auf unterschiedliche Weise bestimmt, was in dem Design der Pipeline durch die zwei unterschiedlichen Segmentierungsfilter berücksichtigt wird (s. Abschn. 3.3.2).

Wahl des Hypothesenpunktes

In der Initialisierungsphase wird als Hypothesnpunkt p_h der Mittelpunkt der, durch die statische Bind- bzw. Initialisierung-Pose vorgegebenen, Handregion (*BLOB*) gewählt (s. Abb. 3.26a). Somit kann p_h während der Initialisierungsphase als konstant und korrekt betrachtet werden. Im Segmentierungsschritt der Trackingphase ist dagegen nicht vorab klar, an welcher Position des Eingabebildes sich die Hand aktuell befindet. So muss p_h hier auf Basis der vorherigen Erkennungsergebnisse angenommen werden. Zur Vereinfachung wird in dieser Arbeit eine langsame, kontinuierlich observierbare globale Handbewegung vorausgesetzt und einfach der Schwerpunkt $(\bar{x}, \bar{y})$ der, für die im vorigen Schritt gefundene, Parameterkonstellation gerenderten Handregion direkt als p_h übernommen (s. Abb. 3.26b). Der Schwerpunkt lässt sich dabei aus den geometrischen Momenten (Meisel, 2012) der vorherigen Handregion berechnen. Eine bessere Abschätzung[19] von p_h könnte allerdings durch ein zusätzliches Tracking erreicht werden.



(a) Hypothesenpunkt (blau) in der Initialisierungsphase. Dieser ist durch den vorab bekannten Mittelpunkt der Bind-Pose gegeben.

(b) Hypothesenpunkt (grün) in der Trackingphase. Dieser wird durch den berechneten Schwerpunkt der Ergebnis-Handregion des vorangehenden Erkennungschritts repräsentiert.

Abbildung 3.26: Wahl eines Hypothesenpunktes zur Segmentierung.

[19] . . . und das sicherlich mit geringerem Aufwand.

Suche des Floodfill Seed Point

Steht der Hypothesenpunkt p_h fest, kann der *Seed Point* p_0 für den *Floodfill*-Algorithmus aus der Menge der hautfarbenen Pixel bestimmt werden. Dies lässt sich über eine lokale Suche im Bereich um die vorgegebenen Koordinaten von p_h realisieren. Die Suche soll dabei einen Pixel ergeben, der einen minimalen Abstand zu p_h liefert und dessen Richtung möglichst der Bewegungsrichtung der Hand entspricht.
Zur Bestimmung der Bewegungsrichtung wird daher zunächst die vektorielle Richtung von einem in einem vorherigen Erkennungsschritt ermittelten Hypothesenpunkt p_{h-1} zu p_h bestimmt. Diese Richtung wird dann zur Identifizierung eines Startpixel innerhalb der 8er-Pixel-Nach-barschaft von p_h verwendet. Ist kein vorheriger Hypothesenpunkt vorhanden, wird der Startpixel zufällig gewählt. Ausgehend von dem Startpixel wird dann sequenziell innerhalb der Nachbarschaft von p_h nach einem passenden Hautfarbenpixel gesucht. Wird dort keiner gefunden, wird der Radius des Suchbereichs inkrementiert und dort wiederum nach einem Hautfarbenpixel geforscht usw. (s. Abb. 3.27).

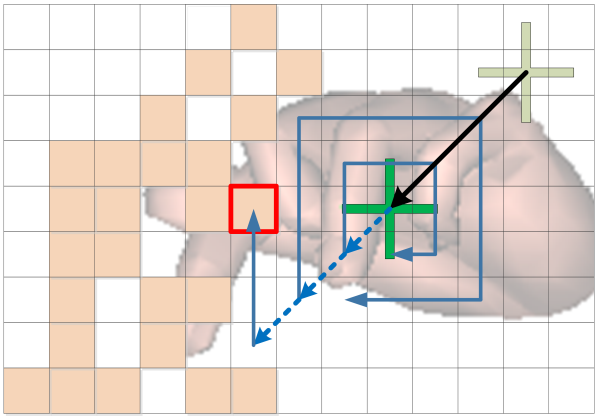


Abbildung 3.27: Seed Point Suche für die Segmentierung. Die Richtung zwischen einem vorangehenden Hypothesenpunkt (blasses grünes Kreuz) und dem aktuell innerhalb der Handregion des vorherigen Schrittes befindlichen Hypothesenpunk p_0 (grünes Kreuz) wird genutzt, um einen Startpixel in der 8er-Nachbarschaft von p_0 zu identifizieren, von dem ausgehend reihum ein Hautfarbenpixel gesucht (rot umrandeter Pixel) wird.

Zur Vermeidung von Fehldetektionen, wird dabei nur bis zu einem maximalen Radius [20] gesucht, sodass nicht jeder Pixel des gesamten Bildes durchlaufen wird und das Risiko der Auswahl eines Hautfarbenpixel eines anderen Körperteils oder eines falsch eingestuften Hautpixels sinkt. Wird innerhalb der Radiusgrenzen kein passender Pixel gefunden, wird die Suche abgebrochen und die Pipeline wechselt in den *Track-Lost-Zustand*. Wird ein Pixel im Suchbereich gefunden, wird dessen korrespondierender Tiefenwert aus dem Tiefenbild gelesen und zunächst auf ungültige Tiefenwerte überprüft (0-Wert). Anschließend wird dieser mit dem Tiefenwert von

[20] Die ideale Grenze des Suchradius hängt dabei von der Geschwindigkeit der Posenerkennung und der globalen Handbewegung ab.

p_h verglichen. Ist der Tiefenunterschied größer als eine bestimmte Vorgabe, wird der gefundene Pixel verworfen und der nächste Pixel gesucht. Besteht der Tiefenwert jedoch den Test, wird die Koordinate des Pixel schließlich als p_0 übernommen und die eigentliche Freistellung durch den *Floodfill*-Algorithmus kann beginnen.

Floodfill

Ist der *Seed Point* ($p_0 = (\hat{x}, \hat{y})$) gegeben, kann durch den *Floodfill*-Algorithmus ein lokal begrenztes Tiefen-Thresholding beschrieben werden, bei dem einzelne Tiefenpixel, die ein vorgegebenes Homogenitätskriterium erfüllen, zu einer Handregion zusammengefasst werden. Zur Bestimmung der Homogenität eines Pixels werden folgende Berechnungen vorgenommen:

$$d(x,y) = |x - \hat{y}| + |y - \hat{y}| \tag{3.7}$$

$$e(x,y) = |Z(x,y) - Z(\hat{x},\hat{y})| \tag{3.8}$$

$d(u,v)$ (s. Gl. 3.7) stellt dabei die *Manhattan-Distanz* (Tönnies, 2005) des aktuell betrachteten Pixel $p_{x,y}$ des Eingabetiefenbildes Z zum Initialpunkt p_0 dar. $e(x,y)$ misst den absoluten Unterschied zwischen der Tiefe an $p_{x,y}$ und p_0 (s. Gl. 3.8). Damit lässt sich das Homogenitätskriterium aufstellen und das segmentierte Bild S berechnen (s. Gl. 3.9)

$$S(x,y) = \begin{cases} Z(x,y) & \text{für} \quad d(x,y) < \tau_d \ \wedge \ e(x,y) < \tau_e \\ 0 & \text{sonst} \end{cases} \tag{3.9}$$

τ_d und τ_e (s. Gl. 3.9) sind anpassbare *Threshold*-Werte für die Entfernung eines Pixels und deren Tiefenunterschied zum Initialpunkt. τ_d wird dabei in dieser Lösung auf die maximale Länge der Hand (bei ausgestreckten Fingern ist das die Entfernung vom Handgelenk zum Mittelfinger) gesetzt. Für τ_e, wird ein Wert[21] von 30 genutzt.

Während der Bestimmung der einzelnen Regionspixel werden zudem die Minima und Maxima der 2D-Koordinaten sowie der Tiefe, im Sinne eines Region-umfassenden Quaders (*Bounding Box*), festgehalten. Die durch die Kanten des Quaders beschriebenen Bereichsgrenzen der Handregion können dann zum einfachen Vergleich mit vorab definierten Größenvorgaben herhalten, was zu einer Überprüfung im Sinne – Ist diese Region eine Hand? – genutzt werden kann.

3.3.5 Online Kinematik Modell

Im Gegensatz zum statischen Strukturmodell stellt das Kinematikmodell sich als Klasse mit eigenen Verhalten heraus. So werden im Kinematikmodell nicht nur die zum Skinning notwendige *Bind Shape Matrix* BSM, die *Inverse Bind Matrices* der einzelnen Bones IBM_i, die *Skinning Weight Table* und die Bone-Transformationsmatrizen der *Bind Pose* gespeichert (s.

[21] vergleichbar hoher Wert, um einzelne Tiefenmessfehler auszugleichen

Abschn. 3.2.2), sondern auch die Beweglichkeit der Hand durch ein online zu transformierendes *Bones System* prozedural nachempfunden. Die Transformation der Bones wird dabei durch die Übergabe einer, entsprechend der im Abschnitt 3.2.2 genannten Vorgaben angegebenen, Modellparameter- bzw. DOF-Konstellation angestoßen.

Das Bones System des Kinematikmodells besteht aus einem Array von 21 Bone-Strukturen, die zur lokalen Rotation der einzelnen Knochen verwendet werden. Jede Bone-Struktur speichert dazu die Ursprungs-Transformationsmatrix (bzw. das Koordinatensystem), die seine relative Lage und Position zum Vorgänger-Knoten innerhalb der *Bind Pose* angibt. Zudem erhält die Bone-Struktur noch einen 3D-Vektor. Die Komponenten des Vektors beschreiben dabei, wie weit ein Bone aktuell um die jeweiligen in der Transformationsmatrix angegebenen Ursprungskoordinatenachsen gedreht wurde. Da die Transformationsmatrizen der *Bind Pose* konstant sind, können die Rotationswinkel dazu genutzt werden, um eine absolute Parameterkonstellation darzustellen, was eine Grundvoraussetzung für die eindeutige Beschreibung der Handpose ist.

Die Transformation erfolgt durch einen expliziten Aufruf einer Aktualisierungsfunktion. Dabei wird für jeden Bone aus seiner Ursprungstransformationsmatrix ein lagebeschreibendes *Quaternion*[22] (Bender und Brill, 2003; Cprogramming.com, 2013) q_{i0} abgeleitet. Aus den Rotationswinkeln wird jeweils ein Quaternion (q_{ix},q_{iy},q_{iz}) abgeleitet, das die Rotation um die entsprechende Achse der Ursprungsmatrix beschreibt. Aus den Rotationsquaternionen lässt sich gemäß Gl. 3.10 ein Quaternion q_{ir} berechnen, das die aktualisierte Lage relativ zum übergeordneten Bone beschreibt.

$$q_{ir} = (q_{iz} * (q_{iy} * q_{ix})) * q_{i0} \tag{3.10}$$

Aus dem Quaternion q_{ir} kann so die relative Transformationsmatrix TR_i des Bones berechnet werden. Da für das Skinning des Handmodells die Transformationsmatrix auf physische (Welt) Koordinaten bezogen werden muss, muss TR_i erst in eine entsprechende Matrix TW_i umgewandelt werden. Die Berechnung von TW_i erfolgt dabei rekursiv über einen kompletten Durchlauf der Bones-Hierarchie im Sinne einer *Vorwärts Kinematik* (s. Gl 3.11), wobei sich die Welttransformationsmatrix TW_0 des ersten Bones (Wrist-Bone) direkt aus seiner relativen Matrix TR_0 ergibt:

$$\begin{aligned} TW_0 &= TR_0 \\ TW_i &= TW_{parent(i)} * TR_i \end{aligned} \tag{3.11}$$

Zur globalen Verschiebung des gesamten Bones Systems wird einfach die Translationskomponente der relativen Matrix des *Wrist-Bone* TR_0 auf die übergebenen Parameterwerte gesetzt.
Vor der eigentlichen Transformation der Bones wird eine an das Kinematikmodell übergebene Parameterkonstellation, durch die im Kinematikmodell statisch definierten Domänen-Constraints, validiert. Dies geschieht, wie in Abschnitt 3.2.2 beschrieben, durch eine einfache Bereichsüberprüfung der einzelnen Parameter. Liegt ein Parameter außerhalb des gültigen Bereichs, wird er

[22] Hier werden Quaternionen verwendet, um die Rotation möglichst schnell und ohne Gefahr eines Gimbel-Lock (vgl. http://www.youtube.com/), der häufig das Resultat einer Euler-Winkel-Rotation ist, zu berechnen.

(ohne einen Fehler auszulösen) auf die nächste Bereichsgrenze gesetzt.

3.3.6 Skinning Transformation, Modellprojektion und Rendering

Die Transformation der Modellstruktur gemäß der durch das Kinematikmodell reflektierten aktuellen Skelett-Konfiguration und deren *Rendering* erfolgt in dieser Arbeit GPU-basiert durch entsprechende Shader-Programme. Der dafür nötige Transfer der statischen Modelldaten des Struktur- und Kinematikmodells in den Grafikspeicher erfolgt dabei zu Beginn der Handposenerkennung durch einen entsprechenden Funktionsaufruf der verantwortlichen Klasse, wobei diese Daten dort bis zur Beendigung in einem jeweiligen *Buffer* (Microsoft, 2013a) verweilen.

Skinning Transformation

Die Transformation der Handstruktur anhand der, durch die 21 Bones des Kinematik-Modells beschriebenen, Skelett-Konfiguration, erfolgt durch eine gängige *Skinning*-Transformation (Barnes und Finch, 2008) innerhalb eines Vertex-Shader. Dabei wird jeder jeder Vertex $\vec{v}^j$ der Handstruktur wie folgt an die, durch die aktuelle Lage der einzelnen Bones, angepasst:

$$\vec{v'}^j = f(TW_i) = \sum_{i=1}^{2} 1(w_i^j * (TW_i * IBM_i * (BSM * \vec{v}^j))) \tag{3.12}$$

Die Bind Shape Matrix BSM und die inversen *Bind Matrices* IBM_i werden dabei vorab ausmultipliziert und vom Kinematikmodell in einen Constant Buffer im Grafikspeicher hinterlegt. Die einzelnen Vertices $\vec{v}^j$ des Strukturmodells werden dagegen zu Beginn der Posenerkennung zusammen mit ihren dazugehörigen, für die einzelnen Bones geltenden, *Skinning*-Gewichten w_i^j der *Weight Table* , in einem Vertex Buffer platziert, der über die Zeit nicht verändert wird und als Eingabe für das GPU-basierte Rendering des Handmodells gilt. Die Welt-Transformationsmatrizen der einzelnen Bones TW_i werden innerhalb des Kinematikmodells regelmäßig verändert (s. Abschn. 3.3.5). Somit können diese als Parameter der *Skinning*-Transformationsfunktion ($f(TW_i)$) gelten. Die einzelnen Bone-Matrizen werden dabei durch die explizite Aufforderung vom Kinematikmodell in einem *Constant Buffer* mit dynamischen CPU-Zugriffsrechten (Microsoft, 2013a) gespeichert.

Handmodell-Projektion

Um die Aufnahmeprojektion der realen Hand während des Renderings des Handmodells nachzuempfinden, bedarf es vorher einer weiteren Transformation der Strukturvertices durch eine Projektionsmatrix.

Innerhalb der DirectX-Renderpipeline ist eine Projektion von homogenen 3D-Koordinaten bzw. Vertices vorgesehen. D.h. eine Koordinate wird durch 4 Komponenten $((x,y,z,w))$ beschrieben, wobei die Rendering-Hardware zur Projektion implizit alle Komponenten durch die vierte Komponente (w) dividiert. Dies wird meist dazu genutzt, um das Modell einer Zentralprojektion (Bender und Brill, 2003) einer virtuellen Kamera, das eine Division von Objektkoordinaten durch ihre jeweilige Tiefe vorsieht, in Form einer explizit formulierten Projektionstransformation (Microsoft, 2013d), nachzuempfinden. Nach der Projektion wird von der Grafikhardware implizit eine *Viewport*-Transformation durchgeführt (Bender und Brill, 2003). Diese erwartet dabei homogene Eingabekoordinaten, die auf den Intervall $[-1;1]$ normiert sind (Microsoft, 2013d).

In dieser Lösung wird eine Projektionsmatrix aus den im Kalibrierungsschritt ermittelten (s. Abschnitt 3.2.1) und im Kameramodell hinterlegten *Camera Intrinsics* abgeleitet, (s. Gl. 3.13) und innerhalb des Vertex-Shader zur Projektion der *Skinning*-transformierten *Vertices* $\vec{v'}^j$ verwendet.

$$\vec{p}^j = \begin{pmatrix} 2\cdot f_x/vp_{width} & 0 & (2\cdot c_x/vp_{width})-1 & 0 \\ 0 & 2\cdot f_y/vp_{height} & (2\cdot c_y/vp_{height})-1 & 0 \\ 0 & 0 & c & -z_n\cdot c \\ 0 & 0 & 1 & 0 \end{pmatrix} * \vec{v'}^j \tag{3.13}$$

mit:

$$c = \frac{z_f}{z_f - z_n}$$

Dabei sind f_x und f_y die in Pixel angegebenen Brennweiten der Kinect-Tiefenkamera in x bzw. y-Richtung, die jeweils auf die aus der Kalibrierung bestimmten Werte gesetzt werden. vp_{width} und vp_{height} stellen die Breite bzw. die Höhe des Viewport-Bildes dar, das gerendert werden soll. Diese entsprechen innerhalb der Lösung den Größen der VGA-aufgelösten Eingabebilder. $\vec{v'}^j$ stellt das Ergebnis der *Skinning Transformation* dar. c ist ein Faktor, der zum Tiefen-*Clipping* der Objektkoordinaten genutzt wird. So wird eine Objektkoodinate, deren Tiefe außerhalb der Bereichsgrenzen z_f (*Z-Far*) und z_n (*Z-Near*) liegt, nach der Projektion nicht dargestellt. z_f und z_n werden dabei auf die Bereichsgrenzen der Kinect-Tiefenmessung gesetzt (s. Abschn. 3.1.1).

Rendering

Mit der oben beschriebenen *Skinning*-Transformation und der Projektion lässt sich jetzt das Handmodell rendern. Ein Rendervorgang wird dabei durch den expliziten Aufruf einer Funktion im Observationsmodell gestartet. Dabei lässt sich vorab einstellen, ob die Ausgabe der bildlichen Modelldarstellung zu Testzwecken in ein Fenster oder standardmäßig in eine Textur im Grafikspeicher erfolgen soll. Abb. 3.28 zeigt die Transformations- und Rendering-Ergebnisse für innerhalb der gültigen Bereichsgrenzen zufällig erzeugte Parameterkonstellationen.

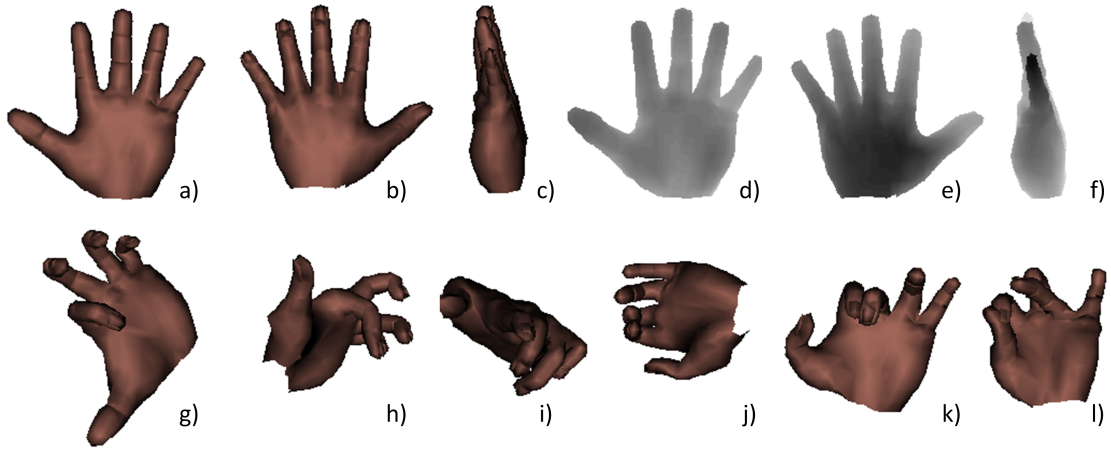


Abbildung 3.28: Rendering des Handmodells. a) bis c): Darstellung des Modells in der Bind Pose von vorne, hinten und der Seite (gerendert durch einen einfachen Beleuchtungs-Pixel-Shader). d) bis f): Zu a) bis c) korrespondierendes Rendering von Tiefenbildern. Hierfür werden die Tiefenwerte der Modell- *Vertices* verwendet und zur Visualisierung skaliert. g) bis l): Rendering von im Rahmen der gültigen Wertebereiche der Modellparameter zufällig generierten Modellposen. k) und l) zeigen, dass auch Posen erzeugt werden, die keiner realen Grundlage entspringen. Die Verwenung eines aufwändigeren dynamischen *Constrained Model* mit einer zusätzlichen Kollisionsdetektion könnte die Generierung dieser unrealistischen Modellposen verhindern.

3.3.7 Observationsmodell

Innerhalb des Observationsmodells erfolgt die Bewertung des Unterschieds zwischen einer aktuellen Modellkonfiguration und einer gegebenen segmentierten Handregion des Tiefenbildes im Sinne einer Fehlermessung. Da die *Particle Swarm Optimierung* eine häufige, für jeden Partikel des Schwarms mehrfach ausgeführte, Fehlermessung erfordert, muss das Observationsmodell sehr effizient implementiert werden. In dieser Arbeit wird das Observationsmodell weitgehend durch individuelle Vertex-, Pixel- und Geometry-Shader-Programme (Microsoft, 2013a) beschrieben, die dediziert durch die GPU ausgeführt werden. Die Messung des Fehlers für eine gegebene Modellparameterkonstellation erfolgt hier in mehreren GPU-Verarbeitungsdurchläufen (*Passes*), wobei in jedem *Pass* eine gewählte Shadersequenz zur Ausführung kommt. Die Aufgabe der Observationsmodell-Klasse liegt so im Wesentlichen darin, für die, durch einen Pass beschriebene, GPU-Verarbeitung die entsprechende Shadersequenz bereit- bzw. einzustellen.
Ausgangspunkt für die Fehlermessung ist eine, an das Observationsmodell übergebene, Modellparameterkonstellation $\vec{p_m}$ (meist in Form einer, durch einen Partikel repräsentierten, Tracking-Hypothese). Diese wird zunächst an das Kinematikmodell weitergereicht und führt dort zu ei-

ner, wie in Abschnitt 3.3.5 beschriebenen, Transformation des Handskeletts bzw. *des Bones System*. Für die so beschriebene Modellpose wird nun durch ein, wie im Abschnitt 3.3.6 geschildertes Rendering, ein Modell-Tiefenbild R erzeugt. Aus den in R gesetzten Modellpixeln r_i ($r_i = (x_{ri}, y_{ri}, z_{ri})$) und der, durch die segmentierte Handregion des Tiefenbild O beschriebenen, Observationspixel o_i ($o_i = (x_{oi}, y_{oi}, z_{oi})$) lässt sich dann der absolute Unterschied d_i zwischen den r_i- und o_i-Pixeln berechnen und in einem Distanzbild D speichern. Durch eine Fehlerfunktion $f(O,R,D)$ kann schließlich der skalare Fehler e zwischen der Modellkonfiguration und der aktuellen Observation ermittelt werden.

Messung der absoluten Pixelunterschiede

Als Parameter für die Fehlerfunktion dient das Distanzbild D, das in jedem seiner Pixel ein Maß d_i für die absoluten Pixel-Unterschied (*Per Pixel Error*) zwischen den einzelnen Pixeln der Modelldarstellung r_i und korrespondierender[23] Observationspixel o_i des Eingabetiefenbildes enthält. d_i ergibt sich dabei im idelaen Fall durch den direkten Unterschied der Tiefe z_{ri} des Modellpixels r_i und der Tiefe z_{oi} des Observationspixel o_i (s. erster Fall in Gl. 3.14).

$$d_i = \begin{cases} |z_r i - z_o i| & \text{für} \quad z_r i > 0 \ \wedge \ z_o i > 0 \\ \sqrt{(x_{ri}-x_{oj})^2 + (y_{ri}-y_{oj})^2 + (z_{ri}-z_{oj})^2} & \text{für} \quad z_{ri} > 0 \ \wedge \ z_{oi} = 0 \\ \sqrt{(x_{rj}-x_{oi})^2 + (y_{rj}-y_{oi})^2 + (z_{rj}-z_{oi})^2} & \text{für} \quad z_{ri} = 0 \ \wedge \ z_{oi} > 0 \\ 0 & \text{sonst} \end{cases} \tag{3.14}$$

Da nicht immer garantiert ist, dass für den Modellpixel r_i ein korrespondierender, an der selben Position befindlicher, Observationspixel o_i gegeben ist und andersherum genauso, muss in diesen Fällen ein anderes Maß für d_i gefunden werden. So wird für den Fall, dass für r_i kein korrespondierender Pixel o_i bereit steht (s. zweiter Fall in Gl. 3.14), wie von Hamer u. a. (2009) vorgeschlagen, zur Bestimmung von d_i der Pixel o_j der Observation O herangezogen, der die minimalste Pixel-Distanz zu r_i aufweist. Zur Berechnung von d_i wird dann nicht mehr nur der absolute Tiefenunterschied verwendet. Stattdessen werden hier r_i und o_j durch die Kamerakalibrierungsdaten in ihre physischen Koordinaten ($(\tilde{x_{ri}}\tilde{y_{ri}}, z_{ri})u.(\tilde{x_{oi}}, \tilde{y_{oi}}, z_{oi})$) zurück transformiert und die Euklidische Distanz zwischen dem Koordinatenpaar als d_i übernommen. Für den Fall, dass o_i gegeben ist, aber kein Pixel r_i bereit steht, wird analog der nächste Modellpixel r_j bestimmt und d_i entsprechend berechnet (s. dritter Fall in Gl. 3.14).
Durch diese Behandlung der Sonderfälle kann die Berechnung der Unterschiede als eine Distanztransformation physischer Koordinaten betrachtet werden (Hamer u. a., 2009), in der die einzelnen d_i die Qualität eines Modellpixels immer durch deren realen Abstand zu seinem nächsten gelegenen Observationspixel bewerten.

[23] Korrespondenz im idealen Fall: Gleiche Pixelposition von r_i und o_i

Fehlerfunktion

Sind die absoluten Pixelunterschiede bekannt, kann die Berechnung des skaleren Fehlerwertes e durch die Fehlerfunktion $f(O,R,D)$ erfolgen. Hierfür wird die von Oikonomidis u. a. (2011b) vorgeschlagene Funktion in angepasster Form verwendet (s. Gl. 3.15).

$$e = f(O,R,D) = \frac{1}{\sigma} \cdot \left(\frac{\sum_{i=1}^{n} \min(d_i, \delta)}{N + \varepsilon} \right) + \lambda \cdot \frac{N_{missed}}{N + \varepsilon} \tag{3.15}$$

mit

$$N = \sum_{i=1}^{n} (o_i \vee r_i) \tag{3.16}$$

$$N_{missed} = \sum_{i=1}^{n} (o_i \vee r_i) - \sum_{i=1}^{n} (o_i \wedge r_i) \tag{3.17}$$

Der erste Term der Fehlerfunktion in Gl. 3.15 stellt die Berechnung des Durchschnitts aller absoluten Pixelunterschiede dar. Um bessere Optimierungsergebnisse zu erreichen, wird während der Aufsummierung der einzelnen Pixelunterschiede deren Wert auf eine maximale Abweichung δ begrenzt. Durch die Anpassung von δ lassen sich dabei einzelne hohe Pixelunterschiede, einer sich ansonsten sehr ähnelnden Modell- und Handpose, die durch Störungen im Aufnahme- und/oder Modellbild entstehen können und zu einem erhöhten durchschnittlichen Fehler führen würden, nivellieren (vgl. Oikonomidis u. a., 2011b). Innerhalb der Fehlerfunktion gibt N (s. Gl. 3.16) die gesamte Anzahl der gesetzten Pixel (Summe der Modellpixel- und Observationspixelanzahl, deren Tiefenwert ungleich 0 ist) an. Durch die Addition des kleinen Wertes ε im Nenner der Durchschnittsberechnung, wird eine Division durch 0 verhindert, falls kein Pixel gesetzt wurde[24]. Weiterhin erfolgt im ersten Term der Funktion eine Normalisierung des berechneten Durchschnittswertes durch den Faktor σ. Dieser kann dabei auf den maximal möglichen Pixelunterschied oder δ gesetzt werden.

Durch den zweiten Term der Fehlerfunktion können die aufgetretenen Spezialfälle während der Berechnung der Pixelunterschiede (s. Abschn. 3.3.7), in denen für einen Modellpixel r_i (bzw. einem Observationspixel o_i) durch Fehlen eines korrespondierenden Observationspixel o_i bzw. (eines Modellpixel r_i) der Unterschied nicht direkt berechnet werden kann, zusätzlich positiv oder negativ bewertet werden. Dazu wird die Anzahl der Pixel N_{missed}, für die diese Umstände gelten, aus der Differenz der gesamten Anzahl der gesetzten Pixel N und der Anzahl der Pixel, für die ein korrespondierender Pixel vorhanden ist, ermittelt (s. Gl. 3.17) und anschließend deren relativer Anteil zu N berechnet. Die Bewertung kann dann durch den über den Intervall $[0,1]$ definierten Gewichtungsparameter λ gesteuert werden. Auf eine weitere Normalisierung der gesamten Fehlerfunktion wird dabei verzichtet, sodass hier, je nach gewähltem λ, Fehlerwerte zwischen 0 und 2 möglich sind.

[24] eher unwahrscheinlich, aber sicher ist sicher

Fehlerauswertung

Die komplette GPU-basierte Auswertung der Fehlerfunktion bleibt in dieser Arbeit offen. So erfolgt hier eine CPU-basierte Distanztransformation des segmentierten Observationsbildes O und des gerendertem (s. Abschn. 3.3.6) Tiefenbild R . Dazu wird eine angepasste Variante des von Rosenfeld und Pfaltz (1966) beschriebenen sequenziellen Algorithmus verwendet, wobei jeder Pixel der resultierenden Bilder (bzw. der Lookup-Tables) M_o und M_r für die der Ursprungstiefenwert 0 war, neben der eher unwichtigen Schachbrettdistanz bzw. (8er-Nachbarschaftsdistanz) die x- und y-Koordinate sowie die Tiefe des naheliegensten Pixel mit definierter Tiefe enthält.

Aus dem gegebenen Observationsbild, dem gerenderten Modellbild und deren Distanztransformierten lassen sich jetzt die einzelnen Pixelunterschiede durch einen *Pixel Shader* innerhalb der GPU berechnen und in einer Ergebnistextur D^0 speichern. Zur GPU-basierten Auswertung der Fehlerfunktion gilt es insbesondere, die sequenziellen Anteile der Fehlerfunktion (z.B. Summenbildung) auf die parallele Verarbeitungsarchitektur[25] abzubilden. Hier wird dies in Form einer sequenziellen Multiskalenstrategie[26] berücksichtigt. Als Ausgangslage dienen dabei die berechnete Textur D^0 sowie O- und R. Innerhalb der Bilder werden jeweils 4er-Pixelregionen betrachtet, deren Informationen zu einem einzigen Pixel zusammengefasst werden. Dazu wird im ersten Schritt (*Pass*) der Sequenz innerhalb eines Pixel-Shader die Summe der Regionspixel aus D^0 berechnet. Zudem wird die Anzahl der übereinstimmenden Pixel N und die der nichtübereinstimmenden Pixel N_{missed} aus O und R innerhalb der Pixelregion ermittelt. Die Summe, N und N_{missed} werden schließlich gemeinsam innerhalb eines einzelnen Pixel einer entsprechenden geringer aufgelösten Ausgabetextur D^1 hinterlegt. Die Textur dient dann als Eingabe für den nächsten Verarbeitungsschritt, in dem wiederum eine Zusammenfassung der entsprechenden niedriger aufgelösten 4er-Regionen vornimmt. Dieses Vorgehen wird wiederholt, bis D^i eine Auflösung von 20x15 Pixel[27] aufweist (also 5 Passes). Die restlichen Pixel werden sequenziell CPU-basiert zusammengefasst.
Aus den so berechneten Teilergebnissen in Form der Summe der Pixelunterschiede, deren Anzahl N und der Anzahl der nicht korrespondierenden Pixelunterschiede N_{missed} lässt sich dann der Fehler durch die definierte Funktion (s. Abschnitt 3.3.7) berechnen.

3.3.8 Initialisierung

Wie bereits angesprochen erfolgt im Rahmen dieser Arbeit eine sehr einfache Initialisierung, die später prinzipiell durch ein aufwändiger zu realisierendes *Single View Pose Estimation*-Verfahren ersetzt werden kann (s. Abschnitt 2.4). So setzt die Initialisierung voraus, dass der Benutzer seine Hand an eine im Ausgabefenster angezeigte Position bewegt, wobei die Handpose der Bind-Pose des Modells entspricht. Die segmentierte Handregion des Tiefenbildes und die gegebene statische Modellparameterkonstellation ($\vec{p_m} = \vec{0}$) der Bind-Pose werden jetzt ein-

[25] Der sequenzielle Durchlauf durch einen einzelnen Shader wäre hier viel zu langsam.
[26] Vergleichbar mit der Gauß-Pyramide
[27] Ab einer bestimmten Größe lohnt sich die Verarbeitung durch Shader nicht mehr, da sie einzelnd betrachtet wesentlich langsamer arbeiten als die CPU.

fach zur direkten Auswertung der Fehlerfunktion (s. vorheriger Abschnitt 3.3.7) verwendet. Eine Übereinstimmung ist dann erreicht, wenn der Fehler e kleiner als ein konstanter Threshold Θ ist.

3.3.9 Tracking durch Particle Swarm Optimization

Wurde innerhalb der Initialisierungsphase eine Hand erkannt, wechselt die Erkennungs-Pipeline in die Tracking-Phase. Dort kommt der im Wesentlichen in Abschnitt 3.3.1 beschriebene *Particle Swarm Optimization Algorithmus* in seiner Ursprungsform zum Einsatz. Lediglich die Generierung der Partikel wird dabei zur Berücksichtigung der temporalen Kontinuität angepasst.

Die Erzeugung der einzelnen Partikel, die jeweils eine hypothetische Modellparameterkonstellation repräsentieren, erfolgt dabei, mit Ausnahme des ersten Partikel des Schwarms $\vec{p_t^0}$, durch Initialisierung der Modellparameter mit Zufallswerten. Dabei wird allgemein sicher gestellt, dass keine doppelten Parameterkonstellation auftreten. Zur Generierung des ersten Partikels $\vec{p_t^0}$ wird diesem direkt die gefundene Parameterkonstellation des letzten Tracking-Schrittes bzw. der Initialisierung $\vec{p}_{t-1}$ zugewiesen.

$$\vec{p}_t^0 = \vec{p}_{t-1} \tag{3.18}$$

Da sich die Lage der Hand und deren lokale Konfiguration der Hand sehr schnell ändern kann (z.B. Fingerbewegung), wird hierfür keine temporale Kontinuität angenommen. So werden die für diese Bewegung verantwortlichen Modellparameter x_t^l als Zufallsvariablen betrachtet, deren Werte sich über den durch die statischen Constraints beschriebenen Wertebereich einheitlich verteilen. Zur Initialisierung eines lokalen Parameters wird also ein, innerhalb seiner gültigen Wertegrenzen c_{max}^l und c_{min}^l erzeugter, Zufallswert verwendet.

$$x_t^l = RNG : c_{min}^{\leq} RNG \leq c_{max}^l \tag{3.19}$$

Für die Bewegungsparameter, die die globale Position der Hand im Raum beschreiben, wird jedoch eine temporale Kontinuität angenommen. So werden diese in einem festen Bereich um die, im letzten Tracking-Schritt ermittelte, Position $\vec{y}_{t-1}$ uniform verteilt[28] Die Bereichsgrenzen der Verteilung werden dabei durch den *Radius* η festgelegt. Um sicherzustellen, dass die zufällig generierte Position $\vec{y}_{t-1}$ im sichtbaren Bildbereich der Kamera liegt, wird diese anhand der Kalibrierungsdaten in den Bildbereich projiziert und dort mit den 2D-Bildgrenzen verglichen.

$$\vec{y_t} = \vec{y_{t-1}} + (RNG_x, RNG_y, RNG_z) : ||\vec{y}_t - \vec{y}_{t-1}|| \leq \eta \ \wedge \ (0,0) \leq proj(\vec{y}_t) < (640,480) \tag{3.20}$$

[28] Keine Normalverteilung, da sich einzelne Ausreißer in weiterer Entfernung zu $\vec{x_{t-1}}$ befinden können.

3.4 Zusammenfassung

In diesem Kapitel wurde die Realisierung der modellbasierten Handposenerkennung diskutiert. Dazu wurde die Realisierung aus den Perspektiven einer *offline* Vorbereitungs- und Modellierungsphase sowie der eigentlichen *online* Posenerkennung unter Berücksichtigung einiger Systemvoraussetzungen und einer Hardwareauswahl (s. Abschn. 3.1) betrachtet. Als Aufnahmegerät wurde dabei der multimodale Kinect-Sensor von Microsoft mit der entsprechenden softwareseitigen Unterstützung in Form des *Kinect for Windows SDK* (Microsoft, 2013b) ausgewählt und im Abschn. 3.1.1 getestet.

Im Rahmen der im Abschnitt (**?**) dokumentierten *offline* Vorbereitungs- und Modellierungsphase, erfolgte dann eine Definition der für die Handposenerkennung nötigen statischen Modelldaten. Dazu zählte insbesondere die Erstellung des Handmodells (s. Abschn. (**?**)) durch das 3D-Authorentool *Autodesk 3DS Max*. *Autodesk 3DS Max* diente dabei im Wesentlichen zur Beschreibung der statischen Modellstruktur (s. Abschn. 3.2.2), eines Handskeletts und der Zusammenhänge zwischen Skelett (s. Abschn. 3.2.2) und Modellstruktur (s. Abschn. 3.2.2). Weiterhin erfolgte im Abschnitt 3.2.1 eine Kamerakalibrierung sowie die Erstellung eines einfachen statistischen Hautfarbenmodells im Abschnitt 3.2.3.

Im online Abschnitt 3.3 des Kapitels wird auf die eigentliche Implementierung der Handposenerkennung eingegangen. Als Basis für das Erkennungssystem dient dabei der *Particle Swarm Optimization Algorithmus* (PSO), der im Detail in Abschnitt 3.3.1 analysiert wurde. Der Algorithmus lässt sich, wie von (Oikonomidis u. a., 2011b) beschrieben, zu einem *Tracking* der Hand ohne explizite Berücksichtigung der Handdynamik einsetzen.
Aufbauend auf der generellen Struktur der PSO, wurde im Abschnitt *sec:impl:online:pipeline* eine Erkennungspipline mit einem dazugehörigen Framework entworfen, welches die einzelnen Schritte der Pipeline und die nötigen Modelle als einzelne Komponenten enthält (s. Abschnitt 3.3.3).

Im restlichen Teil des *online* Abschnittes kann schließlich die, durch die einzelnen Pipeline-Schritte und durch die *online* Modelle definierte, Funktionalität der Handposenerkennung nachvollzogen werden.
Dabei wurde zunächst eine Segmentierung realisiert (s. Abschnitt 3.3.4), die mit Hilfe des Hautfarbenmodells und unter schwacher Berücksichtigung der Handdynamik eine Handregion aus dem Tiefenbild freistellt.

Aufbauend auf den in der offline-Phase ermittelten Modelldaten, wurde im Rahmen von Abschnitt 3.3.5) ein online Kinematikmodell realisiert, das eine parametrisierte Bewegung eines virtuellen Skelettes ermöglicht.
Im Abschnitt 3.3.6 wurde das generell GPU-basierte *Rendering* eines Modelltiefenbildes mit vorangehender *Skinning Transformation* und Projektion der Strukturmodelldaten durch entsprechende Shader-Komponenten umgesetzt.

Zur Messung des Fehlers zwischen Modell und Eingabebild wurde, aufbauend auf den Erkennt-

nissen von Hamer u. a. (2009) und Oikonomidis u. a. (2011b) innerhalb eines GPU-basierten Observationsmodells eine Fehlerfunktion definiert, die eine direkte Messung der pixelbasierten Unterschiede zwischen einem gerenderten Modell und der Observation mit einer Distanztransformation im physischen Raum kombiniert (s. Abschn. 3.3.7). Zur Bildung der Distanztransformation wurden in dieser Arbeit die Eigenschaften der CPU zur sequenziellen Verarbeitung benötigt.

Schließlich erfolgte im Rahmen von Abschnitt 3.3.8, wie bereits angesprochen, eine sehr einfache Umsetzung einer Initialisierung. Zur Umsetzung des Tracking, s. Abschnitt 3.3.9, wird dagegen der PSO-Algorithmus in seiner reinen Form genutzt. Lediglich die Schwarm- bzw. Partikelgenerierung erfolgte in dieser Arbeit durch eine angepasste Variante, bei der die einzelnen Constraints der Modellparameter und die temporale Kontinuität zur Bildung der Partikelhypothesen genutzt wurden.

4 Schluss

Ausgehend vom allgemeinen Forschungsbereich *Human Computer Interaction* (*HCI*) setzt sich diese Bachelorarbeit ausführlich mit dem Problemfeld der der visuellen Handposenerkennung auseinander.

Da die visuelle Handposenerkennung bereits Schwerpunkt vieler Arbeiten ist, erfolgte im Rahmen dieser Arbeit zunächst eine intensive Analyse bereits bestehender Literatur (s. Kapitel 2). Dabei wurde, ausgehend von (Erol u. a., 2007), grundsätzlich zwischen Verfahren zur partiellen Posenbestimmung, sowie den *Single View Pose Estimation* und modellbasierten Verfahren aus dem Bereich der *Full DOF Hand Pose Estimation* unterschieden. Letztere haben dabei eine komplette Bestimmung der realen Handkonfiguration zu jeder Zeit zum Ziel. Prominente Lösungen der Gegenwart wie *Leap Motion*[1] zeigen, dass partielle Verfahren sich durchaus für eine Handposenerkennung unter bestimmten Voraussetzungen eignen. Allerdings bleibt es dabei fraglich, ob eine Posenerkennung nicht für den allgemeinen Fall realisiert werden sollte, wie es die *Single View Pose Estimation* und modellbasierte Verfahren anstreben.

Im Rahmen dieser Arbeit wurde eine einfache modellbasierte Posenerkennung umgesetzt. Ausgangslage für die Realisierung der Posenerkennung war die Analyse vergleichbarer Lösungen im Abschnitt 2.5. Als Eingabegerät war dabei vorab die Verwendung der Microsoft Kinect vorgesehen. Inspiriert durch Oikonomidis u. a. (2011b) wurde dann die *Particle Swarm Optimization* (PSO) als Tracking-Verfahren gewählt. Bei der Wahl des Handmodells wurde sich für ein *Polygon Mesh* entschieden, da hier davon ausgegangen wurde, dass dieses genauere Messungen der Übereinstimmung liefen würde. Zudem wurde ein Observationsmodell aus den Lösungen von (Hamer u. a., 2009) und Oikonomidis u. a. (2011b) abgeleitet (s. Abschnitt 3.3.5).

Zur Erstellung des *Polgon-Mesh*-basierten Handmodells bedurfte es in dieser Arbeit einer intensiveren *offline*-Modellierungsphase (s. Abschnitt 3.2). Neben der statischen Struktur des Handmodells wurden in dieser offline Phase auch die statischen Komponenten eines Kinematikmodells beschrieben (s. Abschnitt 3.2.2), die in der online-Phase eine dynamische Transformation der Handstruktur durch ein lineares Skinning erlaubten.
In der offline-Phase wurde neben der Erstellung der Handmodelldaten zudem eine Kalibrierung des Kinect-Sensors (s. Abschnitt 3.2.1) vorgenommen und ein statistisches Hautfarbenmodell (s. Abschnitt 3.2.3) erstellt, das in der online-Phase zur Segmentierung der Hand genutzt wurde.

Auf Basis der, im offline-Teil erstellten, Modelle erfolgte im online-Teil der Lösung die eigentliche

[1] www.leapmotion.com

Erkennung von Posen. So wurden der *Particle Swarm Optimization Algorithmus* (s. Abschnitt 3.3.1), die notwendigen Schritte und verwendeten dynamischen Modelle in dem dazugehörigen online Abschnitt 3.3 der Arbeit beschrieben. Während der Realisierung wurden einzelne Modelle und die, zur Erkennung notwendigen, Schritte als Komponenten in einem Framework gekapselt (s. Abschnitt 3.3.3). Durch die Komponenten wurde insgesamt eine Pipeline beschrieben (s. Abschnitt 3.3.2), die eine kontiniurliche Erkennung der Posen in den grundsätzlichen Phasen der Vorverarbeitung, der Initialisierung und des Trackings ermöglichte.

Da bei einem Partikel-basierten Tracking-Verfahren die mehrfache Überprüfung von einzelnen, durch die Partikel repräsentierten, Modellparameterhypothesen vorgesehen ist, kann diese als Flaschenhals einer modellbasierten Lösung aufgefasst werden, mit dem die Performance einer Lösung steigt oder fällt. In dieser Arbeit wird dieser Umstand durch eine GPU-basierte Überprüfung der Hypothesen innerhalb des Observationsmodells berücksichtigt (s. Abschnitt 3.3.7)

4.1 Bewertung

Allgemine Performancesteigerung: Grundsätzlich ist die maximal erreichbare Performance der Lösung durch die verwendete Hardware begrenzt. Dies gilt insbesondere für die Bildwechselfrequenz der Kinect-Tiefenmessung von 30 Hz. Pixel-basierte Bildoperationen, wie die Hautfarbenfilterung, ließen sich ebenfalls sehr effektiv durch entsprechende Shader realisieren. Allerdings sollte, aufgrund des langsamen Datentransfer zwischen Hauptspeicher und Grafikspeicher, darauf geachtet werden, die Anzahl der Aktualisierungen einzelner GPU-Ressourcen durch die CPU möglichst gering zu halten.

Segmentierung: Die in dieser Arbeit beschriebene Segmentierung stellt den Versuch dar, die eher mehrdeutige Detektion der Hand anhand der Hautfarbe durch ein zusätzliches einfaches Tracking zu präzisieren. Das beschriebene Verfahren ist dabei sehr experimentell und sollte eher durch ein etabliertes Skin-Object-Tracking-Verfahren ersetzt werden. Zudem lässt sich das gewählte Hautfarbenmodell durchaus weiter optimieren, sodass für die Segmentierung direkt die Hautfarbenpixel auf das Tiefenbild bezogen werden können.

Constraints: Die in dieser Arbeit pro Modellparameter angegebenen Constraints reichen nicht aus, um die gesamte Konfiguration des Modells auf real observierbare Vorgaben zu beschränken. Dies kann zu einer langsamen Konvergenz des Trackings und damit zu erhöhten Laufzeiten führen. So sollte ein aufwändigeres Constraint-Modell verwendet werden, bei dem sämtliche systematischen Abhängigkeiten zwischen den einzelne Modellparameter erfasst und zu einer dynamischen Begrenzung der Parameterwerte genutzt werden.

Observationsmodell: Die GPU-Auswertung des Anpassungsfehlers innerhalb des Observationsmodells erforderte in dieser Arbeit noch eine CPU-basierte Distanztransformation. Dies sollte aufgrund der häufigen Konsultation des Observationsmodell durch die PSO oder im Allgemein bei einem Multi-Hypothesen-Tracking vermieden werden. Zur weiteren Steigerung der Geschwindigkeit der Fehlermessung, sollte die in dieser Lösung nur

für jeweils eine Modellhypothese beschriebene GPU-basierte Überprüfung zu einem Tile-basierten Ansatz erweitert werden, um mehrere Hypothesen auf einmal überprüfen zu können.

4.2 Ausblick

Mit zunehmendem Bedarf an natürlichen Schnittstellen wird auch die Handposenerkennung ein probates Mittel zur Human Computer Interaction bleiben. Die rasante Entwicklung paralleler Hardware-Architekturen und entsprechender Verabeitungs-Algorithmen lässt dabei den Schluss zu, dass ein modellbasierter Ansatz in naher Zukunft ähnlich gute Erkennungsraten wie ein partielles Verfahren erreicht.
Die Entscheidung zwischen einem dieser Verfahren könnte dabei zu Gunsten des modellbasierten Ansatzes ausfallen, da sich hiermit die Dynamik der Hand allgemeiner und intuitiver nachvollziehen lässt. Zudem passt dieser Ansatz optimal zum Verarbeitungsschema gängiger interaktiver Anwendungen aus dem Bereich Virtual-, Augmented- sowie Mixed-Reality und – last but not least– des *Gaming*. So kann davon ausgegangen werden, dass in naher Zukunft, in immer interaktiver und immersiver werdenden Anwendungen modellbasierte Verfahren einen wesentlichen Beitrag dazu leisten, die Hand und deren Bewegung in eine virtuelle Umgebung einzubinden.

Dennoch wird immer das Problem der Initialisierung der Schwachpunkt eines auf einem grafischen Modell basierenden Ansatzes bleiben. So eignet sich die statische Struktur eines einzelnen Modells nicht oder nur wenig, um die natürlichen Variationen der Handform zwischen unterschiedlichen Personen von vornherein zu beschreiben. Hierfür bedarf es einer Modellkalibrierung, im Sinne einer Anpassung des Modells an die Hand des Benutzers, auf die in der Literatur relativ selten eingegangen wird. Dies führt zu der Annahme, dass weiterhin die Entwicklung des ultimativen Handdetektors im Sinne einer Single View Pose Estimation ein Forschungsziel bleiben wird.

Grundsätzlich wird davon ausgegangen, dass sich die Forschungsarbeit im Bereich der visuellen Handposenerkennung langfristig auf die Interaktion zwischen zwei Händen (Oikonomidis u. a., 2012), anderen hautfarbenen Körperteilen und Objekten der Umgebung (Hamer u. a., 2009; Oikonomidis u. a., 2011a) fokussiert. Die Ähnlichkeit der Hautfarbe und das hohe Maß an zusätzlichen Verdeckungsfällen erschweren dabei die Erkennung ungemein und erfordern besondere Lösungsstrategien.

Literaturverzeichnis

[Albrecht u. a. 2003] ALBRECHT, Irene ; HABER, Jörg ; SEIDEL, Hans-Peter: Construction and animation of anatomically based human hand models. In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville and Switzerland and Switzerland : Eurographics Association, 2003 (SCA '03), S. 98–109. – URL http://dl.acm.org/citation.cfm?id=846276.846290. – ISBN 1-58113-659-5

[Alon u. a. 2009] ALON, Jonathan ; ATHITSOS, V. ; QUAN YUAN ; SCLAROFF, S.: A Unified Framework for Gesture Recognition and Spatiotemporal Gesture Segmentation. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31 (2009), Nr. 9, S. 1685–1699. – ISSN 0162-8828

[Argyros und Lourakis 2004] ARGYROS, Antonis A. ; LOURAKIS, Manolis I. A.: Real-Time Tracking of Multiple Skin-Colored Objects with a Possibly Moving Camera. In: *In: ECCV*, 2004, S. 368–379

[Athitsos und Sclaroff 2003] ATHITSOS, Vassilis ; SCLAROFF, Stan: Estimating 3D Hand Pose from a Cluttered Image. In: *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 2 (2003), S. 432–432–9 vol.2. – URL http://dx.doi.org/10.1109/cvpr.2003.1211500. – ISSN 1063-6919

[Autodesk 2013a] AUTODESK: *3DS MAX*. 2013. – URL http://www.autodesk.de/products/autodesk-3ds-max/overview. – Zugriffsdatum: 16.09.2013

[Autodesk 2013b] AUTODESK ; AUTODESK (Hrsg.): *Autodesk 3ds Max Help*. 2013. – URL http://www.autodesk.com/3dsmax-help-v2011. – Zugriffsdatum: 25.09.2013

[Autodesk 2013c] AUTODESK: *MAXScript Reference*. 2013. – URL http://docs.autodesk.com/3DSMAX/14/ENU/MAXScript%20Help%202012/. – Zugriffsdatum: 16.09.2013

[Barnes und Finch 2008] BARNES, Mark ; FINCH, Ellen L. ; THE KHRONOS GROUP INC. SONY COMPUTER ENTERTAINMENT INC (Hrsg.): *COLLADA 1.5.0 Specification*. 2008. – URL http://www.khronos.org/files/collada_spec_1_5.pdf. – Zugriffsdatum: 26.09.2013

[Ben Henia u. a. 2010] BEN HENIA, Ouissem ; HARITI, Mohamed ; BOUAKAZ, Saida: A two-step minimization algorithm for model-based hand tracking. In: *18th International Conference on Computer Graphics, Visualization and Computer Vision (WSCG)*, URL http://liris.cnrs.fr/publis/?id=4574, 2010

[Bender und Brill 2003] BENDER, Michael ; BRILL, Manfred: *Computergrafik: Ein anwendungsorientiertes Lehrbuch ; [Website mit Quellcodes, Lösungen zu den Aufgaben, Einführungen in die Tools und mehr]*. München and Wien : Hanser, 2003. – ISBN 3-446-22150-6

[Bradski und Kaehler 2008] BRADSKI, Gary R. ; KAEHLER, Adrian: *Learning OpenCV: Computer vision with the OpenCV library*. 1. Sebastopol and CA : O'Reilly, 2008. – ISBN 0596516134

[Bray u. a. 2004] BRAY, M. ; KOLLER-MEIER, E. ; MULLER, P. ; GOOL, L. van ; SCHRAUDOLPH, N.N: 3D hand tracking by rapid stochastic gradient descent using a skinning model. In: *Visual Media Production, 2004. (CVMP). 1st European Conference on*, 2004, S. 59–68

[Bretzner u. a. 2002] BRETZNER, L. ; LAPTEV, I. ; LINDEBERG, T.: Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering. In: *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, 2002, S. 423–428

[Breuer 2005] BREUER, Pia: *Entwicklung einer prototypischen Gestenerkennung in Echtzeit unter Verwendung einer IR Tiefenkamera*. Koblenz and GE, Universität Koblenz Landau, Dissertation, 2005

[Brown und Capson 2012] BROWN, J.A ; CAPSON, D.W: A Framework for 3D Model-Based Visual Tracking Using a GPU-Accelerated Particle Filter. In: *Visualization and Computer Graphics, IEEE Transactions on, title=A Framework for 3D Model-Based Visual Tracking Using a GPU-Accelerated Particle Filter* 18 (2012), Nr. 1, S. 68–80. – ISSN 1077-2626

[Burros 2013] BURROS, Nicolas: *Kinect Calibration*. 2013. – URL `http://nicolas.burrus.name/index.php/Research/KinectCalibration`. – Zugriffsdatum: 22.09.2013

[Buschmann 1996] BUSCHMANN, Frank: *Pattern-oriented software architecture: A system of patterns*. Chichester and New York : Wiley, 1996. – ISBN 9780471958697

[Castaneda und Navab 2011] CASTANEDA, Victor ; NAVAB, Nassir: *Time-of-Flight and Kinect Imaging*. 2011. – URL `http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors_LabCourse_Kinect.pdf`. – Zugriffsdatum: 15.09.2013

[Causo u. a. 2008] CAUSO, A. ; UEDA, E. ; KURITA, Y. ; MATSUMOTO, Y. ; OGASAWARA, T.: Model-based hand pose estimation using multiple viewpoint silhouette images and Unscented Kalman Filter. In: *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, 2008, S. 291–296

[Chan Wah Ng und Surendra Ranganath 2002] CHAN WAH NG ; SURENDRA RANGANATH: Real-time gesture recognition system and application. In: *Image and Vision Computing* 20 (2002), Nr. 13–14, S. 993–1007. – URL `http://www.sciencedirect.com/science/article/pii/S0262885602001130`. – ISSN 0262-8856

[Clerc und Kennedy 2002] CLERC, M. ; KENNEDY, J.: The particle swarm - explosion, stability, and convergence in a multidimensional complex space. In: *Evolutionary Computation, IEEE Transactions on* 6 (2002), Nr. 1, S. 58–73. – ISSN 1089-778X

[Cprogramming.com 2013] CPROGRAMMING.COM ; CPROGRAMMING.COM (Hrsg.): *Using Quaternion to Perform 3D rotations*. 2013. – URL http://www.cprogramming.com/tutorial/3d/quaternions.html. – Zugriffsdatum: 30.09.2013

[collada dom.sourceforge.net 2013] DOM.SOURCEFORGE.NET collada ; DOM.SOURCEFORGE.NET collada (Hrsg.): *COLLADA Document Object Model*. 2013. – URL http://collada-dom.sourceforge.net/. – Zugriffsdatum: 16.09.2013

[Eberhart und Kennedy 1995] EBERHART, R. ; KENNEDY, J.: A new optimizer using particle swarm theory. In: *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, 1995, S. 39–43

[Erol u. a. 2007] EROL, Ali ; BEBIS, George ; NICOLESCU, Mircea ; BOYLE, Richard D. ; TWOMBLY, Xander: Vision-based hand pose estimation: A review. In: *Comput. Vis. Image Underst.* 108 (2007), Nr. 1-2, S. 52–73. – URL http://dx.doi.org/10.1016/j.cviu.2006.10.012. – ISSN 1077-3142

[Freedman u. a. 2010] FREEDMAN, Barak ; SHPUNT, Alexander ; MACHLINE, Meir ; YOEL, Arieli: *Depth mapping using projected patterns*. 2010. – URL http://www.freepatentsonline.com/y2010/0118123.html

[Gang Hu und Gao 2011] GANG HU ; GAO, Q.: Gesture Analysis Using 3D Camera, Shape Features and Particle Filters. In: *Computer and Robot Vision (CRV), 2011 Canadian Conference on*, 2011, S. 204–211

[Ghosh u. a. 2010] GHOSH, Soumita ; ZHENG, Jianmin ; CHEN, Wenyu ; ZHANG, Jane ; CAI, Yiyu: Real-time 3D markerless multiple hand detection and tracking for human computer interaction applications. In: *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry*. New York and NY and USA : ACM, 2010 (VRCAI '10), S. 323–330. – URL http://doi.acm.org/10.1145/1900179.1900247. – ISBN 978-1-4503-0459-7

[Hamer u. a. 2009] HAMER, H. ; SCHINDLER, K. ; KOLLER-MEIER, E. ; GOOL, L. van: Tracking a hand manipulating an object. In: *Computer Vision, 2009 IEEE 12th International Conference on*, 2009, S. 1475–1482

[Hui Liang u. a. 2012] HUI LIANG ; JUNSONG YUAN ; THALMANN, Daniel: 3D fingertip and palm tracking in depth image sequences. In: *Proceedings of the 20th ACM international conference on Multimedia*. New York and NY and USA : ACM, 2012 (MM '12), S. 785–788. – URL http://doi.acm.org/10.1145/2393347.2396312. – ISBN 978-1-4503-1089-5

[Jones und Rehg 1999] JONES, M.J ; REHG, J.M: Statistical color models with application to skin detection. In: *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on* Bd. 1, 1999, S. –280 Vol. 1

[Kennedy und Eberhart 1995] KENNEDY, J. ; EBERHART, R.: Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on* Bd. 4, 1995, S. 1942–1948 vol.4

[Keskin u. a. 2011] KESKIN, C. ; KIRAC, F. ; KARA, Y.E ; AKARUN, L.: Real time hand pose estimation using depth sensors. In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, 2011, S. 1228–1234

[Khronos Group 2013] KHRONOS GROUP: *COLLADA - Digital Asset and FX Exchange Schema*. 2013. – URL http://www.khronos.org/collada/. – Zugriffsdatum: 23.09.2013

[Kolsch und Turk 2004] KOLSCH, M. ; TURK, M.: Fast 2D Hand Tracking with Flocks of Features and Multi-Cue Integration. In: *Computer Vision and Pattern Recognition Workshop on Real-Time Vision for Human-Computer Interaction*, 2004, S. 158

[La Gorce u. a. 2011] LA GORCE, M. d. ; FLEET, D.J ; PARAGIOS, N.: Model-Based 3D Hand Pose Estimation from Monocular Video. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on, title=Model-Based 3D Hand Pose Estimation from Monocular Video* 33 (2011), Nr. 9, S. 1793–1805. – ISSN 0162-8828

[Meisel 2012] MEISEL, Andreas: *Robot Vision: Skript*. 2012. – Skript

[Microsoft 2013a] MICROSOFT ; MICROSOFT DEVELOPER NETWORK (MSDN) (Hrsg.): *DirectX Graphics and Gaming*. 2013. – URL http://msdn.microsoft.com/en-us/library/ee663274%28v=vs.85%29.aspx. – Zugriffsdatum: 16.09.2013

[Microsoft 2013b] MICROSOFT ; MICROSOFT CORPORATION (Hrsg.): *Kinect for Windows*. 2013. – URL http://www.microsoft.com/en-us/kinectforwindows/. – Zugriffsdatum: 15.09.2013

[Microsoft 2013c] MICROSOFT ; MICROSOFT (Hrsg.): *Kinect for Windows SDK*. 2013. – URL http://msdn.microsoft.com/en-us/library/hh855347.aspx. – Zugriffsdatum: 17.09.2013

[Microsoft 2013d] MICROSOFT: *The Direct3D Transformation Pipeline*. 2013. – URL http://msdn.microsoft.com/en-us/library/windows/desktop/ee418867%28v=vs.85%29.aspx. – Zugriffsdatum: 21.10.2013

[msdn.microsoft.com 2013] MSDN.MICROSOFT.COM ; MICROSOFT (Hrsg.): *Kinect for Windows SDK Documentation*. 2013. – URL http://msdn.microsoft.com/en-us/library/hh855347.aspx. – Zugriffsdatum: 16.09.2013

[Nguyen u. a. 2011] NGUYEN, Thi Thanh M. ; PHAM, Ngoc H. ; DONG, Thai van ; NGUYEN, Viet S. ; TRAN, Thi Thanh H.: A fully automatic hand gesture recognition system for human-robot interaction. In: *Proceedings of the Second Symposium on Information and Communication Technology*. New York and NY and USA : ACM, 2011 (SoICT '11), S. 112–119. – URL http://doi.acm.org/10.1145/2069216.2069241. – ISBN 978-1-4503-0880-9

[Oikonomidis u. a. 2011a] OIKONOMIDIS, I. ; KYRIAZIS, N. ; ARGYROS, A.A: Full DOF tracking of a hand interacting with an object by modeling occlusions and physical constraints. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, S. 2088–2095

[Oikonomidis u. a. 2012] OIKONOMIDIS, I. ; KYRIAZIS, N. ; ARGYROS, A.A: Tracking the articulated motion of two strongly interacting hands. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, S. 1862–1869

[Oikonomidis u. a. 2011b] OIKONOMIDIS, Iason ; KYRIAZIS, Nikolaos ; ARGYROS, Antonis A.: Efficient model-based 3D tracking of hand articulations using Kinect. In: *Proceedings of the British Machine Vision Conference*, BMVA Press, 2011, S. 101.1–101.11. – ISBN 1-901725-43-X

[Oikonomidis u. a. 2011c] OIKONOMIDIS, Iasonas ; KYRIAZIS, Nikolaos ; ARGYROS, Antonis A.: Markerless and efficient 26-DOF hand pose recovery. In: *Proceedings of the 10th Asian conference on Computer vision - Volume Part III*. Berlin and Heidelberg : Springer-Verlag, 2011 (ACCV'10), S. 744–757. – URL http://dl.acm.org/citation.cfm?id=1966049.1966108. – ISBN 978-3-642-19317-0

[opencv.org 2013] OPENCV.ORG ; OPENCV.ORG (Hrsg.): *OpenCV*. 2013. – URL http://opencv.org/. – Zugriffsdatum: 16.09.2013

[Petersen 2013] PETERSEN, Iwer: *Using object tracking for dynamic video projection mapping*. Hamburg, Hochschule für Angewandte Wissenschaften, Dissertation, 2013

[Phung u. a. 2005] PHUNG, Son L. ; BOUZERDOUM, Abdesselam ; CHAI, Douglas: Skin Segmentation Using Color Pixel Classification: Analysis and Comparison. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27 (2005), Nr. 1, S. 148–154. – URL http://dx.doi.org/10.1109/TPAMI.2005.17. – ISSN 0162-8828

[Ren u. a. 2011] REN, Zhou ; YUAN, Junsong ; ZHANG, Zhengyou: Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In: *Proceedings of the 19th ACM international conference on Multimedia*. New York and NY and USA : ACM, 2011 (MM '11), S. 1093–1096. – URL http://doi.acm.org/10.1145/2072298.2071946. – ISBN 978-1-4503-0616-4

[Romero u. a. 2009] ROMERO, J. ; KJELLSTROM, H. ; KRAGIC, D.: Monocular real-time 3D articulated hand pose estimation. In: *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, 2009, S. 87–92

[Rosales u. a. 2001] ROSALES, Romer ; ATHITSOS, Vassilis ; SIGAL, Leonid ; SCLAROFF, Stan: 3D Hand Pose Reconstruction Using Specialized Mappings. In: *In ICCV*, 2001, S. 378–385

[Rosenfeld und Pfaltz 1966] ROSENFELD, Azriel ; PFALTZ, John L.: Sequential Operations in Digital Picture Processing. In: *J. ACM* 13 (1966), Nr. 4, S. 471–494. – URL http://doi.acm.org/10.1145/321356.321357. – ISSN 0004-5411

[Sajjawiso und Kanongchaiyos 2011] SAJJAWISO, T. ; KANONGCHAIYOS, P.: 3D Hand pose modeling from uncalibrate monocular images. In: *Computer Science and Software Engineering (JCSSE), 2011 Eighth International Joint Conference on*, 2011, S. 177–181

[Shimada u. a. 2001] SHIMADA, N. ; KIMURA, K. ; SHIRAI, Y.: Real-time 3D hand posture estimation based on 2D appearance retrieval using monocular camera. In: *Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 2001. Proceedings. IEEE ICCV Workshop on*, 2001, S. 23–30

[Shimada u. a. 1998] SHIMADA, N. ; SHIRAI, Y. ; KUNO, Y. ; MIURA, J.: Hand gesture estimation and model refinement using monocular camera-ambiguity limitation by inequality constraints. In: *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, 1998, S. 268–273

[Shotton u. a. 2011] SHOTTON, J. ; FITZGIBBON, A. ; COOK, M. ; SHARP, T. ; FINOCCHIO, M. ; MOORE, R. ; KIPMAN, A. ; BLAKE, A.: Real-time human pose recognition in parts from single depth images. In: *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*. Washington and DC and USA : IEEE Computer Society, 2011 (CVPR '11), S. 1297–1304. – URL http://dx.doi.org/10.1109/CVPR.2011.5995316. – ISBN 978-1-4577-0394-2

[Stenger u. a. 2001] STENGER, B. ; MENDONCA, P.R.S ; CIPOLLA, R.: Model-based 3D tracking of an articulated hand. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* Bd. 2, 2001, S. II–310–II–315 vol.2

[Stenger u. a. 2006] STENGER, Bjorn ; THAYANANTHAN, Arasanathan ; TORR, Philip H. S. ; CIPOLLA, Roberto: Model-Based Hand Tracking Using a Hierarchical Bayesian Filter. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 28 (2006), Nr. 9, S. 1372–1384. – URL http://dx.doi.org/10.1109/TPAMI.2006.189. – ISSN 0162-8828

[Stenger 2004] STENGER, Björn Dietmar R.: *Model-Based Hand Tracking Using A Hierarchical Bayesian Filter*. Cambridge and UK, University of Cambridge, Dissertation, 2004

[Sudderth u. a. 2004] SUDDERTH, Erik B. ; MANDEL, Michael I. ; FREEMAN, William T. ; WILLSKY, Alan S.: Visual Hand Tracking Using Nonparametric Belief Propagation. In: *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 12 - Volume 12*. Washington and DC and USA : IEEE Computer Society, 2004 (CVPRW '04), S. 189–. – URL http://dl.acm.org/citation.cfm?id=1032643.1033080. – ISBN 0-7695-2158-4

[Tofighi u. a. 2010] TOFIGHI, G. ; MONADJEMI, S.A ; GHASEM-AGHAEE, N.: Rapid hand posture recognition using Adaptive Histogram Template of Skin and hand edge contour. In: *Machine Vision and Image Processing (MVIP), 2010 6th Iranian*, 2010, S. 1–5

[Tönnies 2005] TÖNNIES, Klaus D.: *Grundlagen der Bildverarbeitung*. München and Boston [u.a.] : Pearson Studium, 2005 (Informatik : Bildverarbeitung). – ISBN 3-8273-7155-4

[Viola und Jones 2001] VIOLA, P. ; JONES, M.: Rapid object detection using a boosted cascade of simple features. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* Bd. 1, 2001, S. I–511–I–518 vol.1

[Wagner 2010] WAGNER, Stefan: *Partikelschwarmoptimierung*. 2010. – URL http://www-user.tu-chemnitz.de/~wags/about/particle-swarm-optimization.pdf. – Zugriffsdatum: 17.10.2013

[Wang u. a. 2007] WANG, Xiying ; ZHANG, Xiwen ; DAI, Guozhong: Tracking of deformable human hand in real time as continuous input for gesture-based interaction. In: *Proceedings of the 12th international conference on Intelligent user interfaces*. New York and NY and USA : ACM, 2007 (IUI '07), S. 235–242. – URL http://doi.acm.org/10.1145/1216295.1216338. – ISBN 1-59593-481-2

[Ying Wu und Huang 1999] YING WU ; HUANG, T.S: Capturing articulated human hand motion: a divide-and-conquer approach. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* Bd. 1, 1999, S. 606–611 vol.1

[Yuan Yao und Yun Fu 2012] YUAN YAO ; YUN FU: Real-Time Hand Pose Estimation from RGB-D Sensor. In: *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, 2012, S. 705–710

[Zhigeng Pan u. a. 2010] ZHIGENG PAN ; YANG LI ; MINGMIN ZHANG ; CHAO SUN ; KANGDE GUO ; XING TANG ; ZHOU, S.Z: A real-time multi-cue hand tracking algorithm based on computer vision. In: *Virtual Reality Conference (VR), 2010 IEEE*, 2010, S. 219–222

[Zhu u. a. 2006] ZHU, Zhenhua ; GAO, Shuming ; WAN, Huagen ; YANG, Wenzhen: Trajectory-Based Grasp Interaction for Virtual Environments. In: NISHITA, Tomoyuki (Hrsg.) ; PENG, Qunsheng (Hrsg.) ; SEIDEL, Hans-Peter (Hrsg.): *Advances in Computer Graphics* Bd. 4035. Springer Berlin Heidelberg, 2006, S. 300–311. – URL http://dx.doi.org/10.1007/11784203_26. – ISBN 978-3-540-35638-7

Abbildungsverzeichnis

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. Oktober 2013 Hannes Dieck