Handgestenerkennung mit 3D-Punktwolken

Schriftliche Prüfungsarbeit
für die Master-Prüfung des Studiengangs Angewandte Informatik
an der Ruhr-Universität Bochum

vorgelegt von

Zalecki, Kristof

20.01.2015

1. Prüfer: PD Dr. Rolf Würtz
2. Prüfer: Andreas Nilkens

Inhaltsverzeichnis

1 Einleitung **4**

2 Theorie **6**
2.1 Time-of-Flight Kamera 6
2.1.1 Funktionsweise 6
2.1.2 Besonderheiten 9
2.2 Punktwolken 9
2.3 Graphen 10
2.4 Hauptkomponentenanalyse 11
2.4.1 Maximum Variance Formulation 12

3 k-Nearest Neighbor Algorithmus **15**
3.1 Nearest-Neighbor Suche in Punktwolken 15
3.2 Morton-Order 16
3.3 k-Nearest Neighbor Graph Construction Algorithmus 18
3.4 Parallelisierung 22

4 Matching **23**
4.1 Transformation der Daten 23
4.1.1 Translation 24
4.1.2 Rotation 25
4.1.3 Skalierung 27
4.1.4 Ausrichtung des Graphen 30
4.2 Algorithmus 30

5 Implementierung **34**
5.1 Libraries und Entwicklungsumgebung 34
5.2 Adaf 34
5.2.1 Adaf Framework 35
5.2.2 Adaf SDK 35

6 Testläufe **37**
6.1 Versuchsaufbau 37
6.2 Generierung der Graphen 37
6.3 Die Handgesten 40
6.4 Matchingphase 46
6.5 Testreihen 47
6.5.1 Auswertung - Allgemein 50
6.5.2 Auswertung - Daumen 52
6.5.3 Auswertung - Faust 54

6.5.4 Auswertung - Greifen 57
6.5.5 Auswertung - Hand Hinten 59
6.5.6 Auswertung - Hand Vorne 61
6.5.7 Auswertung - Hand Vorne Finger Zusammen 63
6.5.8 Auswertung - OK 65
6.5.9 Auswertung - Victory 67

7 Fazit **70**

A Anhang **72**

Literatur **84**

1 Einleitung

Die Erkennung von Handgesten spielt in der Unterhaltungselektronik sowie im industriellen und wissenschaftlichen Bereich eine immer größere Rolle. Aufgrund des stetigen Wandels rückt die Steuerung von Endgeräten per Handgeste immer mehr in den Vordergrund. Das ist nicht nur der Tatsache geschuldet, dass die dafür benötigte Hardware immer günstiger in der Produktion wird. Auch der Drang nach immer neueren und einfacheren Methoden bei der Bedienung von elektronischen Geräten treibt diese Entwicklung voran. Viele Hersteller bieten bereits kostengünstige Hardware an, die Entwicklern eine gute Basis bieten.
In der Computer Vision gibt es bereits eine Vielzahl von Algorithmen und Programmen, die sich mit diesem Thema befassen. Trotz der Verfügbarkeit guter Hardware stellt die Handgestenerkennung eine enorm große Herausforderung dar, da im Gegensatz zur Objekterkennung nicht mit starren Gegenständen gearbeitet wird, sondern mit flexiblen Händen, die sich bei jedem Menschen bezüglich Größe und Proportionen stark unterscheiden können. Es kann auch nicht sichergestellt werden, dass ein und dieselbe Handgeste bei jeder Aufnahme identisch ausfällt, was Lage und Form angeht. All diese Faktoren müssen bei der Entwicklung guter Algorithmen berücksichtigt werden. Diese fußen zum Teil auf völlig unterschiedlichen Ansätzen, wobei die Funktionsweise der Kamera eine nicht unwesentliche Rolle spielt. Während 2D-Kameras nur in der Lage sind, eine dreidimensionale Szene auf eine zweidimensionale Fläche abzubilden, und somit zu einem Verlust an Informationen führen, vermögen 3D-Kameras auch die Tiefeninformationen einer betrachteten Szene zu verarbeiten. Das erfordert allerdings auch eine sinnvolle Repräsentation der Daten, um diese für die weitere Verarbeitung nutzen zu können. Eine Möglichkeit besteht zum Beispiel darin, Distanzbilder zu erstellen, die die Tiefeninformationen jedes Pixels mit Hilfe eines einfachen Wertes darstellen. Objekte können dann unter Anderem anhand ihrer Konturen oder Form analysiert und weiter verarbeitet werden.
Eine weitere Möglichkeit ist die Nutzung von Punktwolken, mit denen sich Objekte anhand einer Menge von Einzelpunkten abbilden lassen. Punktwolken kommen bereits bei einer Vielzahl von Anwendungsgebieten zum Einsatz und haben dort bereits ihre Vielseitigkeit unter Beweis gestellt.

Diese Arbeit befasst sich mit der Handgestenerkennung auf Grundlage von 3D-Punktwolken. Das Ziel ist es, schrittweise ein Erkennungsverfahren zu entwickeln, das in der Lage ist, verschiedene Handgesten zu speichern und diese anschließend in den Aufnahmen einer *Time-of-Flight* Kamera wiederzuerkennen. Anschließend soll das Verfahren evaluiert und die Ergebnisse

präsentiert werden, um sich ein Bild von der Qualität des Verfahrens machen zu können und mögliche Vor- und Nachteile benennen zu können.
Im Zuge dessen wird in Kapitel 2 zunächst der theoretische Hintergrund präsentiert, der dieser Arbeit zugrunde liegt und für das weitere Verständnis notwendig ist. Unter anderem wird die verwendete Kamera und ihre Arbeitsweise in Abschnitt 2.1 vorgestellt. Zudem wird in den Abschnitten 2.2 und 2.3 kurz auf Punktwolken und Graphen eingegangen, um dem Leser eine Vorstellung über diese Datenstrukturen geben zu können.
Kapitel 3 befasst sich dann mit dem *k-Nearest Neighbor* Algorithmus, der dazu verwendet wird Graphen aus aufgenommenen Punktwolken zu erstellen. In Abschnitt 3.1 wird als erstes das allgemeine Problem der Graphensuche in Punktwolken vorgestellt, das auf Grundlage des in den Abschnitten 3.2 bis 3.4 im Detail beschriebenen Algorithmus gelöst wird.
Im Anschluss wird in Kapitel 4 der entwickelte Matching Algorithmus vorgestellt, der auf Grundlage dieser aus Punktwolken generierten Graphen Handgesten erkennen soll. Hierfür wird in Abschnitt 4.1 als erstes auf die Vorverarbeitung eingegangen, mit deren Hilfe die Graphen und die zu vergleichende Punktwolke in die richtige Position und Größe für das Matching gebracht werden sollen. Anschließend wird der Algorithmus selber in Abschnitt 4.2 im Detail vorgestellt und erklärt.
Auf die programmatische Umsetzung des Verfahrens wird kurz in Kapitel 5 eingegangen, das zudem auch die verwendete Entwicklungsumgebung behandelt.
Darauf folgend werden in Kapitel 6 Testreihen vorgestellt und evaluiert, die die Qualität und Leistung des Algorithmus untersuchen sollen. Die dafür verwendeten Handgesten werden zunächst in Abschnitt 6.3 präsentiert und deren Testreihen anschließend in Abschnitt 6.5 einzeln ausgewertet.
In einem abschließenden Fazit in Kapitel 7 werden die Vor- und Nachteile der Handgestenerkennung auf Grundlage von 3D-Punktwolken dargestellt. Es wird ebenfalls auf mögliche Verbesserungen eingegangen, die aus den Ergebnissen dieser Arbeit resultieren könnten.

2 Theorie

Das folgende Kapitel beschreibt die mathematischen und theoretischen Grundlagen, auf denen die in dieser Arbeit behandelten Algorithmen zur Handgestenerkennung basieren.
Im Rahmen dessen wird zunächst auf Punktwolken eingegangen, welche die Daten der Kamera liefern. Des weiteren wird auf die Kamera selbst eingegangen, die für die Erzeugung der Daten zuständig ist.

2.1 Time-of-Flight Kamera

Für die Aufnahme der dreidimensionalen Daten kommt eine Time-of-Flight Kamera, kurz TOF-Kamera, der Marke PMD, zum Einsatz. Diese arbeitet mit Infrarotsignalen, im Folgenden als IR-Signale bezeichnet, um die Distanzen einer beobachteten Szene zu berechnen. Dabei enthält jedes Pixel des von der Kamera aufgenommenen Bildes die Tiefeninformationen des dazugehörigen Punktes in der Szene. Die Distanz wird ermittelt, indem Infrarotlicht von der Kamera ausgesendet und von einem, vor der Kamera befindlichen, Objekt wieder zurück geworfen wird. Der Sensor der Kamera misst die Phasenverschiebung des eingehenden Infrarotlichtes und berechnet damit die Distanz. Auf die genaue Funktionsweise der Kamera wird in 2.1.1 eingegangen.
TOF-Kameras können dreidimensionale Szenen erfassen, ohne auf komplexe Computer-Vision Algorithmen zurückgreifen zu müssen. Aufgrund ihres einfachen Aufbaus finden sie bereits in vielen Bereichen Anwendung, wie zum Beispiel der Robotik, der 3D Rekonstruktion und - wie im Rahmen dieser Arbeit erläutert wird - im Bereich der Mensch-Maschine-Interaktion. Ein prominenteres Beispiel für die Anwendung von Infrarotsignalen zur Distanzberechnung ist die Kinect Kamera. Sie hat, als Teil der XBox Spielekonsole, bereits in vielen Haushalten Einzug gehalten, aber auch in wissenschaftlichen Bereichen findet sie Anwendung. Im Gegensatz zu der hier betrachteten TOF-Kamera verwendet die Kinect Kamera jedoch ein anderes Verfahren. Sie projiziert ein Muster aus Infrarotlicht auf eine beobachtete Szene, um mittels Triangulation die Distanzen zu berechnen. [2, S. 1ff.].

2.1.1 Funktionsweise

Wie bereits erwähnt, werden die Distanzen einzelner Punkte zur Kamera anhand von Infrarotsignalen berechnet. Im Detail funktioniert dies folgendermaßen:
Die Infrarot LED der Kamera strahlt ein Infrarotsignal aus, welches von einem davor befindlichen Objekt reflektiert und zurück zur Kamera geworfen

wird. Der Sensor der Kamera fängt das Signal auf und misst die Phasenverschiebung des ausgestrahlten und des zurückgeworfenen Signals und berechnet daraus die Distanz. Abbildung 1 zeigt die verwendete Kamera.

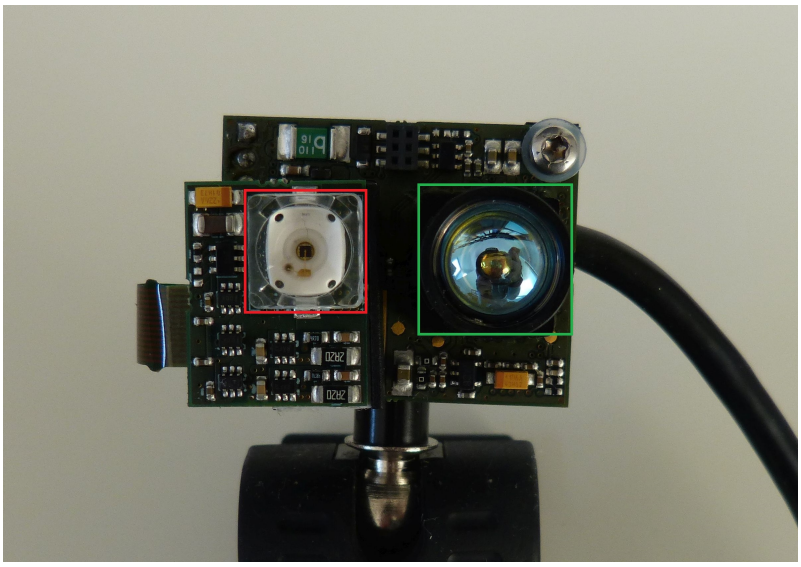

Abbildung 1: Die Time-of-Flight Kamera von PMD: Sie besteht im wesentlichen aus einer LED, die ein Infrarot Signal ausstrahlt (rot umrandet) und einem Sensor, der das zurückgeworfene Infrarotsignal wieder auffängt (grün umrandet).

Wie in Abbildung 2 zu sehen ist sendet die Kamera ein IR-Signal f (hier blau dargestellt) in Richtung des Objekts aus. Das vom Objekt zurückgeworfene IR-Signal (rot dargestellt) weist eine leichte Phasenverschiebung auf und trifft auf den Sensor der Kamera. Die Phasenverschiebung wird anhand der Relation von vier verschiedenen elektrischen Ladungswerten ermittelt, die durch die Kontrollsignale $\{C_1, C_2, C_3, C_4\}$ repräsentiert werden und jeweils eine 90 Grad Phasenverschiebung zueinander haben. Sie geben an, wie viele Elektronen vom Sensor in der jeweiligen Ausrichtung aufgenommen wurden. Aus diesen vier Werten lassen sich, wie in Abbildung 3 dargestellt, vier Quantitätswerte $\{Q_1, Q_2, Q_3, Q_4\}$ ermitteln, welche die Ladungsmenge des jeweiligen Kontrollsignals von C_1 bis C_4 abbilden.

Die Phasenverschiebung berechnet sich anschließend wie folgt aus den vier Quantitätswerten:

$$t_d = \arctan\left(\frac{Q_3 - Q_4}{Q_1 - Q_2}\right). \tag{1}$$

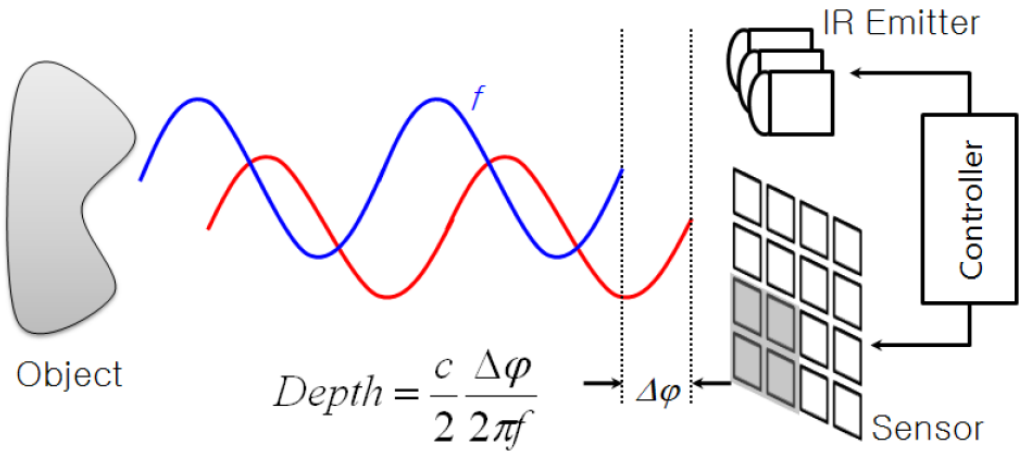

Abbildung 2: Funktionsweise der TOF-Kamera: Anhand der Phasenverschiebung des ausgestrahlten IR Signals (*blau*) und des reflektierten IR Signals (*rot*) wird die Distanz berechnet. [2, S. 2, Abb. 1.1]

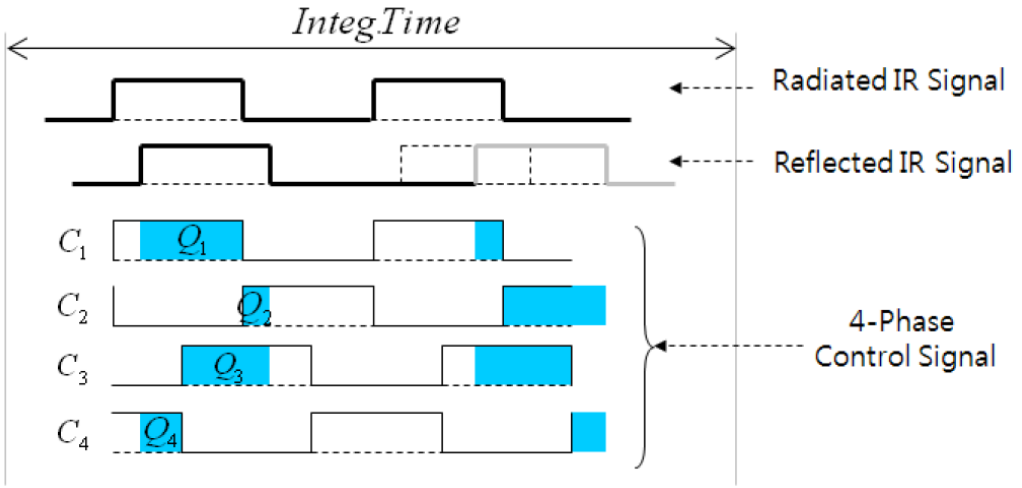

Abbildung 3: Tiefenberechnug mittels des IR Signals. Die aus den vier Kontrollsignalen $\{C_1, C_2, C_3, C_4\}$ ermittelten Quantitätswerte $\{Q_1, Q_2, Q_3, Q_4\}$ erlauben die Berechnung der Phasenverschiebung des IR Signals. [2, S. 2, Abb. 1.2]

Aus dem Wert für die Phasenverschiebung t_d, der Frequenz f des IR Signals und des Werts der Lichtgeschwindigkeit c kann die Distanz d berechnet werden:

$$d = \frac{c}{2f}\frac{t_d}{2\pi}. \tag{2}$$

2.1.2 Besonderheiten

Aufgrund der Funktionsweise der TOF-Kamera stellen sich einige Besonderheiten ein, die die Distanzmessung in einzelnen Szenen beeinflussen können. Wie in 2 gezeigt, hängt die Berechnung der Distanz von der Frequenz f des Infrarot Signals ab. Das hat vor allem zur Folge, dass die Messgenauigkeit nur in einem bestimmten Abstandsintervall zur Kamera funktioniert. Weit entfernte Objekte können zum Beispiel falsch gemessen werden, wenn das Signal um mehr als eine Phase verschoben am Sensor eintrifft. Da aber die Stärke der Infrarot LED begrenzt ist, tritt dieses Problem eher selten auf.
Die Beschaffenheit eines Objekts spielt ebenfalls eine große Rolle. Je nach Farbe und Struktur der Oberfläche kann das auftreffende Infrarotlicht besser oder schlechter reflektiert werden. Während helle und raue Oberflächen vorwiegend gut reflektieren, kann es bei glatten und dunklen Oberflächen zu Problemen führen. Transparente Oberflächen, wie zum Beispiel Glas, werden von der TOF-Kamera so gut wie gar nicht erkannt.
Laut [2] weisen TOF-Kameras jedoch ein noch viel entscheidenderes Problem auf. Durch schnelle Bewegungen der Kamera oder des Objekts kann es zu Bewegungsunschärfe kommen. Wie bereits gezeigt, hängt die Distanzmessung von der Frequenz der Kamera ab. Niedrigere Abtastfrequenzen erlauben eine genauere Distanzmessung, haben jedoch zur Folge, dass das Ergebnis leichter durch Bewegungen beeinflusst wird. Andersherum bietet eine schnellere Abtastfrequenz zwar eine höhere Robustheit bezüglich der Bewegungsunschärfe, führt aber auch zu einer qualitativen Abnahme der Distanzberechnung.
Alles in allem weisen TOF-Kameras zwar einige Schwächen auf, die bei stereoskopischen 3D Kameras nicht auftreten, bieten aber gleichzeitig eine robuste und einfache Möglichkeit, dreidimensionale Daten der Umgebung zu erfassen und zu verarbeiten. Vor allem die direkte Berechnung der Distanzdaten auf der Kamera stellt einen großen Vorteil dar, da an dieser Stelle auf komplexe Algorithmen verzichtet werden kann.

2.2 Punktwolken

Punktwolken - im englischen auch *point clouds* genannnt - werden in der Bildverarbeitung zur Repräsentation zwei- und dreidimensionaler Daten ver-

wendet. In diesem Zusammenhang besteht eine Punktwolke im wesentlichen aus einer Menge ungeordneter Koordinaten, die die Oberfläche eines Objekts abbilden [7]. Allgemeiner gefasst beschränken sich Punktwolken aber nicht nur auf Oberflächen, sondern erlauben es, jeglichen mehrdimensionalen Datensatz zu fassen und im zwei- und dreidimensionalen Fall auch entsprechend zu visualisieren.
Abbildung 4 veranschaulicht beispielhaft eine dreidimensionale Punktwolke aus zwei Perspektiven. Die linke Ansicht zeigt die Punktwolke einer Hand aus einer seitlichen Lage, während die rechte Ansicht die gleiche Punktwolke aus einer frontalen Ansicht zeigt. Vor allem links ist gut zu erkennen, wie die Oberfläche der abgebildeten Hand durch die Punktwolke wiedergegeben wird.

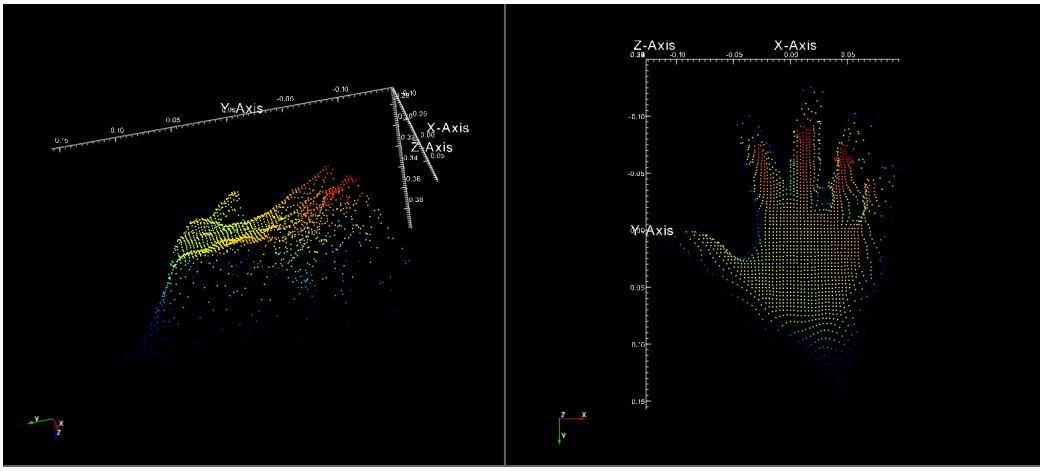

Abbildung 4: Beispiel einer Punktwolke: Zu erkennen ist die Abbildung einer Hand aus zwei Perspektiven. Links aus einer seitlichen Lage und rechts aus einer frontalen Ansicht der Handfläche.

2.3 Graphen

Um Handgesten für das Matching speichern zu können, bedarf es einer sinnvollen Repräsentation, die einerseits die grundlegenden, aber auch die lokalen Strukturen der aufgenommenen Punktwolke erhalten kann und andererseits nur einen Bruchteil der Datenmenge benötigt. Graphen stellen hierfür eine geeignete Wahl dar, da sie genau diese Anforderungen erfüllen.
„*Graphen sind eine kombinatorische Struktur, die bei der Modellierung zahlreicher Probleme Verwendung findet*“ und lassen sich wie folgt definieren [8]:

Definition 1. *Ein Graph G ist ein Tupel (V,E), wobei V eine (endliche) nichtleere Menge von Knoten (engl. vertices) ist. Die Menge E ist eine Teilmenge der zweielementigen Teilmengen von V, also* $E \subseteq \binom{V}{2} := \{\{x, y\} | x, y \in V, x \neq y\}$. *Die Elemente der Menge E bezeichnet man als Kanten (engl. edges).*

Genauer gesagt bezieht sich diese Definition nur auf ungerichtete Graphen, für die gelten muss, dass eine Verbindung, die von einem Knoten x zu einem Knoten y besteht, auch in entgegengesetzter Richtung gegeben sein muss. Abbildung 5 veranschaulicht einen solchen Graphen. Die Knoten a, b, c und d sind über Kanten mit ihren Nachbarknoten verbunden. Dabei können die einzelnen Knoten mehrere Kanten zu verschiedenen Knoten haben.

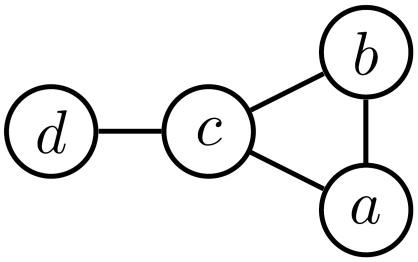

Abbildung 5: Ungerichteter Graph: Die Knoten des Graphen sind über ungerichtete Kanten miteinander verbunden.[8]

Bei den im Rahmen der Handgestenerkennung verwendeten Graphen handelt es sich um solche ungerichteten Graphen, deren Punkte zudem eine räumliche Lage im Dreidimensionalen Raum haben. Verbindungen zwischen einzelnen Punkten beschränken sich nicht nur auf eine Ebene, sondern können sich in alle Richtungen der drei Dimensionen erstrecken. Entscheidend ist auch, dass jede Kante eine spezifische Länge hat, die sich aus der euklidischen Distanz der Koordinaten ihrer zwei verbindenden Punkte ergibt. Auf die Definition eines solchen, aus einer Punktwolke generierten Graphen wird nachfolgend in 3.1 eingegangen.

2.4 Hauptkomponentenanalyse

Bei der Hauptkomponentenanalyse, oder kurz PCA (*principal component analysis*), handelt es sich um ein Verfahren, das unter anderem zur Reduktion der Dimensionalität von Daten verwendet wird, aber auch für die verlustbehaftete Datenkompression und die Extraktion von Merkmalen genutzt wird.

Laut [1] lässt sich die PCA anhand zweier Definitionen wiedergeben, die den gleichen Algorithmus beschreiben:

1. "Die orthogonale Projektion von Daten auf einen niedrigerdimensionalen linearen Raum, auch bekannt als *principal subspace*, sodass die Varianz der projizierten Daten maximiert wird."[3]

2. "Die lineare Projektion, welche die durchschnittlichen Projektionskosten, definiert als der Mittelwert der quadrierten Distanz zweier Datenpunkte und ihrer Projektion, verringert."[6]

Die erste der beiden Definitionen, wird im Folgenden näher betrachtet, da sie beim Matching Verfahren der Graphen eine Rolle spielt. Für eine detailliertere Zusammenfassung der Hauptkomponentenanalyse sei an dieser Stelle auf [1] verwiesen.

2.4.1 Maximum Variance Formulation

Die *Maximum Variance Formulation*, wie in [1] beschrieben, basiert auf der in Abschnitt 2.4, Punkt 1 gegebene Definition der PCA. Ziel dieses Verfahrens ist es, Daten in einen Raum mit Dimensionalität $M < D$ zu projizieren und gleichzeitig die Varianz der Daten zu maximieren.
Sei hierfür $\{\mathbf{x}_n\}$, mit $n = 1, ..., N$ eine Menge an Daten und $\mathbf{x}_n$ eine euklidische Variable mit Dimensionalität D. Diese Daten sollen nun auf einen eindimensionalen Raum ($M = 1$) abgebildet werden. Hierfür lässt sich ein D-dimensionaler Vektor $\mathbf{u}_1$ definieren, der der Einfachheit halber einen Einheitsvektor mit $\mathbf{u}_1^T\mathbf{u}_1 = 1$ bildet. Jeder Punkt $\mathbf{x}_n$ wird nun auf einen skalaren Wert $\mathbf{u}_1^T\mathbf{x}_n$ projiziert. Der Mittelwert der projizierten Daten ist $\mathbf{u}_1^T\overline{\mathbf{x}}_n$ und ergibt sich aus dem Mittelwert $\overline{\mathbf{x}}$ aller Datenpunkte, mit

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n. \tag{3}$$

Zusammen mit der Kovarianz Matrix $\mathbf{S}$, definiert durch

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T \tag{4}$$

ergibt sich die Varianz der Daten, welche gegeben ist durch

$$\frac{1}{N}\sum_{n=1}^{N}\{\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\overline{\mathbf{x}}\}^2 = \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1. \tag{5}$$

Will man nun diese Varianz in Bezug auf $\mathbf{u}_1$ maximieren, muss darauf geachtet werden, dass $\|\mathbf{u}_1\| \to \infty$ nicht eintreten darf. Dies lässt sich erreichen, indem ein Lagrange Multiplikator eingeführt wird, bezeichnet durch λ_1, der eine Begrenzung ermöglicht. Es ergibt sich folgender für die Maximierung geeigneter Term:

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^T\mathbf{u}_1), \tag{6}$$

dessen Ableitung bezüglich $\mathbf{u}_1$ auf Null gesetzt und aufgelöst zu folgender Gleichung führt:

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 = \lambda_1. \tag{7}$$

Daraus ergibt sich, dass die Varianz maximal wird, wenn $\mathbf{u}_1$ dem Eigenvektor von $\mathbf{S}$ entspricht, welcher den Eigenwert λ_1 hat. Der Eigenvektor stellt die erste Hauptkomponente der PCA dar - auch bekannt als *principal component*.

Jede weitere Hauptkomponente berechnet sich nun inkrementell aus der größten Varianz, die orthogonal zur bereits gefundenen Hauptkomponente gefunden werden kann. Genauer gesagt ergeben sich die für eine Abbildung D-dimensionaler Daten auf einen M-dimensionalen Raum die Hauptkomponenten aus den Eigenwerten $\lambda_1, ..., \lambda_M$ der Kovarianzmatrix $\mathbf{S}$.

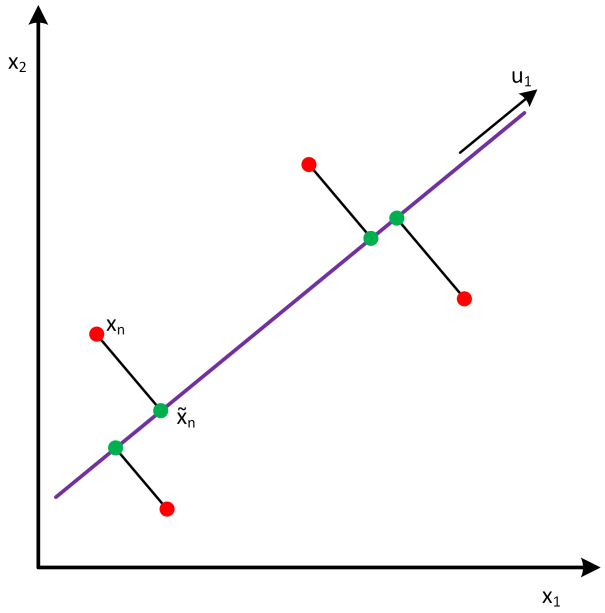

Abbildung 6: Veranschaulichung einer PCA: Die Datenpunkte (rot) werden orthogonal auf eine niedrigere Dimensionalität (dargestellt durch die violettfarbene Linie) abgebildet (grüne Punkte), wodurch ihre Varianz maximiert wird. Die Richtung in der diese Linie verläuft bildet die erste Hauptkomponente, die der Richtung von Vektor u_1 entspricht.

3 k-Nearest Neighbor Algorithmus

Die Wahl eines geeigneten *Nearest-Neighbor* Algorithmus stellt bei der Generierung der Graphen den wichtigsten Punkt dar. Hauptaugenmerk liegt einerseits auf der Laufzeit des Algorithmus, da die spätere Verarbeitung der Daten möglichst nah an der Frequenz der verwendeten TOF-Kamera liegen soll, andererseits auf der Approximation an ein optimales Ergebnis, um brauchbare Graphen für das Graph Matching zu erhalten. Im Rahmen dessen fiel die Wahl auf den *k-Nearest Neighbor Graph Construction* Algortithmus von *M. Connor* und *P. Kumar* [4]. Dieser zeichnet sich vor allem durch eine gute Parallelisierbarkeit aus, sowie die Tatsache, dass er speziell für Punktwolken entwickelt wurde.
Dieses Kapitel setzt sich im Detail mit dem Algorithmus auseinander und gibt einen Überblick über seine Eigenschaften. Dafür wird im Folgenden zunächst allgemein auf die Auffindung von Nachbarschaften in Punktwolken eingegangen, um einen Einblick in die Problemstellung zu bekommen und um eine Grundlage für die weiteren Abschnitte zu erhalten. Im Anschluss wird der im Rahmen dieser Arbeit verwendete Algorithmus im Detail vorgestellt. Dabei wird unter anderem auch auf Besonderheiten, wie zum Beispiel *Z-Kurven*, eingegangen. Abschließend wird die Laufzeit des Algorithmus näher betrachtet.

3.1 Nearest-Neighbor Suche in Punktwolken

Um aus einer Punktwolke einen für die weitere Verarbeitung geeigneten Graphen zu generieren, bedarf es eines Algorithmus, welcher es ermöglicht die Nachbarschaften der einzelnen Punkte zu analysieren und deren unmittelbare Nachbarn ausfindig zu machen. Dieses Problem lässt sich dabei wie folgt definieren:
Sei

$$P = \{p_1, p_2, ..., p_n\} \tag{8}$$

eine Punktwolke in $\mathbb{R}^d$ bestehend aus n Punkten, wobei im Rahmen dieser Arbeit $d = 3$ gelte. Für jeden Punkt $p_i \in P$ werden die $k \leq n - 1$ Nachbarn gesucht, welche einen minimalen Abstand zu dem jeweiligen Punkt aufweisen. Diese seien definiert durch $\mathcal{N}_i^k$.
Man erhält einen *k-Nearest Neighbor* Graphen mit Knotenmenge

$$V = \{p_1, p_2, ..., p_n\} \tag{9}$$

und Kantenmenge

$$E = \{(p_i, p_j) : p_i \in \mathcal{N}_j^k \ oder \ p_j \in \mathcal{N}_i^k\}. \tag{10}$$

Hierbei sei zu beachten, dass es sich um einen gerichteten Graphen handelt, da gelten kann, dass $p_i \in \mathcal{N}_j^k$ und $p_j \notin \mathcal{N}_i^k$. Für den speziellen Fall eines ungerichteten Graphen muss für die Kantenmenge gelten

$$E = \{(p_i, p_j) : p_i \in \mathcal{N}_j^k \; und \; p_j \in \mathcal{N}_i^k\}. \tag{11}$$

3.2 Morton-Order

Einen wichtigen Teil des *k-Nearest Neighbor Graph Construction* Algorithmus bilden sogenannte Z-Kurven - auch als *Morton Order* bekannt. Bei Z-Kurven handelt es sich um raumfüllende Kurven, mit denen sich mehrdimensionale räumliche Abhängigkeiten in eindimensionalen Strukturen darstellen lassen. Vor allem die guten nachbarschaftserhaltenden Eigenschaften sind für die Verwendung der Kurven im Algorithmus von Bedeutung. Berechnet werden diese Kurven durch bitweises Verschränken der einzelnen Koordinatenwerte, die für jede Koordinate einen Wert ergeben, anhand dessen seine Beziehung zu seinen Nachbarn festgestellt werden kann.

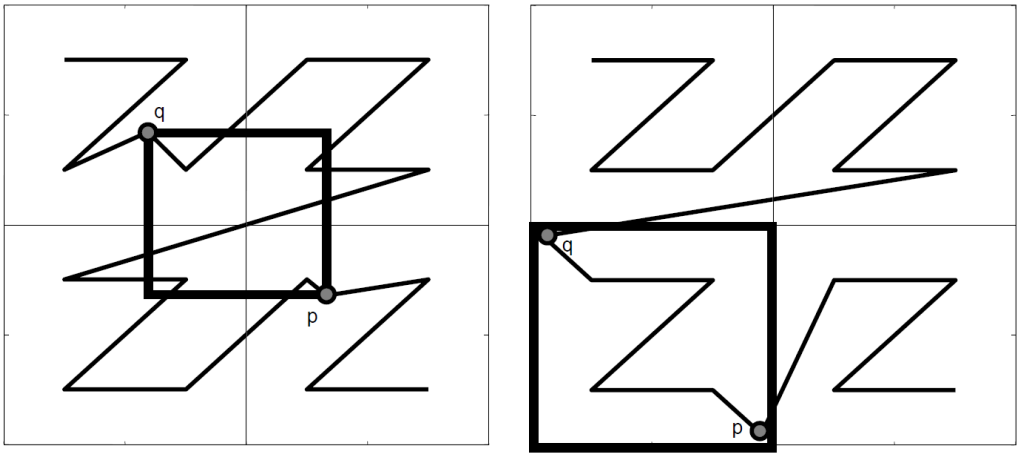

Abbildung 7: *Z-Kurve.*[4]

Alorithmus 1 des *k-Nearest Neighbor Graph Construction* Algorithmus bedient sich zwar des Prinzips der Z-Kurven, vereinfacht dieses jedoch aus Gründen der Effizienzsteigerung. Anstatt die Z-Werte selbst zu berechnen, vergleicht der Algorithmus die binäre Darstellung zweier Koordinatenwerte und sucht ausgehend von der höchsten Stelle die Position heraus, an der sich die beiden Werte unterscheiden. Mit Hilfe dieses Verfahrens kann festgestellt werden, welcher der beiden Werte höher- und welcher geringerwertig ist.

Alorithmus 1 besteht aus den folgenden zwei Prozeduren:

1. COMPARE(**point** p ,**point** q)
2. XOR_{MSB}(**double** p, **double** q)

Die Prozedur COMPARE iteriert über die d (in diesem Fall $d = 3$) Dimensionen der Eingabepunkte p und q und ruft für die zweite Prozedur XOR_{MSB} mit den Werten der jeweiligen Dimension auf (Zeile 4). Gesucht wird hierbei die Dimension, an der sich die beiden Punkte in der höchsten Stelle in Binärdarstellung unterscheiden. Als Ergebnis wird zurückgegeben, ob Punkt p in der gefundenen Dimension größer oder kleiner ist als Punkt q. Die in Zeile 4 aufgerufene Prozedur XOR_{MSB} ist für die Berechnung der höchsten differierenden Stelle in Binärdarstellung zuständig. Sie arbeitet mit Fließkommazahlen als Eingabe und berechnet zunächst deren Exponenten (Zeile 8). Gleichen sich diese, wird daraufhin die Stelle mit dem ersten höherwertigen differierenden Bit berechnet und dessen Differenz zum Exponenten des ersten Wertes zurückgegeben (Zeilen 9-11). Sollten sich die beiden Exponenten der Eingabewerte unterscheiden, wird der jeweils größere Exponent zurückgegeben (Zeilen 12 und 13).

Algorithm 1 Floating Point Morton Order Algorithm

Require: d-dimensional points p and q
Ensure: **true** if $p < q$ in Morton order

```
1:  procedure COMPARE(point p, point q)
2:      x ← 0; dim ← 0
3:      for all j = 0 to d do
4:          y ← XOR_MSB(p_(j), q_(j))
5:          if x < y then
6:              x ← y; dim ← j
        return p_(dim) < q_(dim)

7:  procedure XOR_MSB(double a, double b)
8:      x ← EXPONENT(a); y ← EXPONENT(b)
9:      if x = y then
10:         z ← MSDB(MANTISSA(a), MANTISSA(b))
11:         x ← x − z return x
12:     if y < x then return x
13:     else return y
```

3.3 k-Nearest Neighbor Graph Construction Algorithmus

Der Algorithmus, dargestellt in 2, besteht im wesentlichen aus den folgenden drei Teilen [4]:

1. *Prepocessing Phase*
2. *Sliding Window Phase*
3. *Search Phase*

Im ersten Schritt - der *Preprocessing Phase* - werden die Punkte der Punktwolke in einem eindimensionalen Array sortiert. Als Vergleichsoperator dient hierbei die in 3.2 behandelte *Morton Ordnung* (Zeile 2). Das führt dazu, dass räumlich nah beieinander liegende Punkte im Array ebenfalls nah beieinander angeordnet werden. Als Sortierverfahren wird eine parallelisierte Form von *Quick Sort* verwendet, die vor allem den Vorteil hat, dass direkt auf dem Eingabe-Array gearbeitet wird.

Die beiden Folgeschritte werden ebenfalls parallelisiert ausgeführt. Dies wird dadurch ermöglicht, dass nach dem Sortieren des Arrays keine weiteren Änderungen auf den Daten mehr stattfinden. Es werden lediglich lesende Operationen benötigt, die die Integrität des Arrays nicht beeinflussen.
Für die *Sliding Window Phase* wird zunächst für jeden Punkt p_i in P eine Menge A_i gebildet, die die k nächsten Nachbarn enthält. Hierfür werden jeweils die ck nächsten Nachbarn links und die ck nächsten Nachbarn rechts vom betrachteten Punkt gewählt, wobei c eine Konstante darstellt, welche den Wert $1/2$ nicht unterschreiten darf (Zeile 4). Bei der Implementierung des vorgestellten Algorithmus wurde mit $c = 1$ gearbeitet. Man erhält in diesem Fall eine Menge von $2k$ nächsten Nachbarn für die weitere Verarbeitung. Für jeden Punkt in der gefundenen Menge werden anschließend die Distanzen zum Punkt p_i berechnet, aus denen die k nächstgelegenen Nachbarn ermittelt und in der Menge A_i gespeichert werden. Die erhaltene partielle Lösung bedarf im nächsten Schritt einer Verfeinerung, um sicher zu gehen, dass tatsächlich eine optimale Lösung gefunden wurde.
Stellt man sich die in A_i enthaltene partielle Lösung als einen Ball vor, der die nächsten Nachbarn von Punkt p_i enthält (wie in Abbildung 8 dargestellt), dann gilt es nun zu prüfen, ob die gefundene Lösung tatsächlich innerhalb des betrachteten Intervalls von $\{p_{i-ck}, ..., p_{i+ck}\}$ liegt.
Der in A_i befindliche Punkt mit der größten Distanz und einem größeren Wert in Morton Ordnung als p_i und der Punkt mit der größten Distanz und

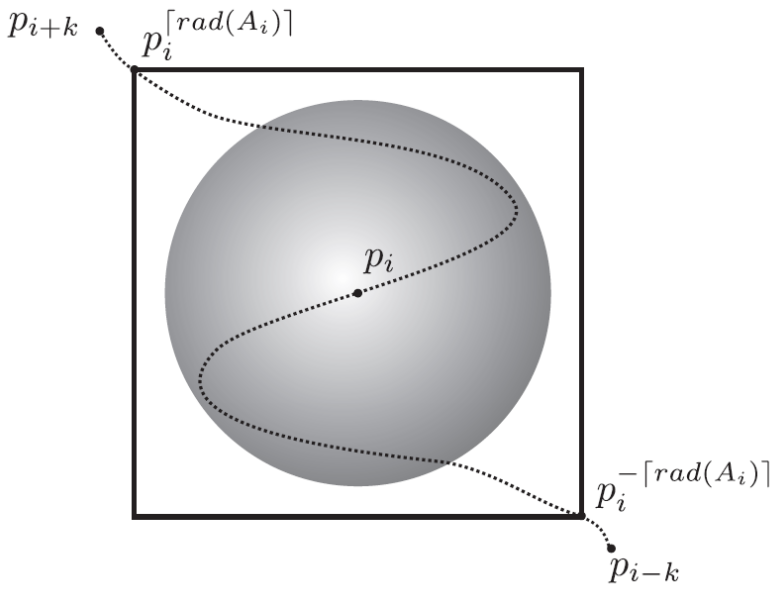

Abbildung 8: Darstellung eines *Nearest Neighbor Ball* für Punkt p_i einer Punktwolke P: Der Ball liegt innerhalb der Obergrenze $p_i^{\lceil rad(A_i)\rceil}$ und der Untergrenze $p_i^{-\lceil rad(A_i)\rceil}$. Die Punkte p_{i+k} und p_{i-k} dürfen im Falle einer optimalen Lösung für das *nearest neighbor Problem* nicht innerhalb dieser Grenzen liegen.

einem geringeren Wert in Morton Ordnung als p_i bilden den Radius des *nearest neighbor Balls* und tragen die Bezeichnungen $p_i^{\lceil rad(A_i)\rceil}$ und $p_i^{-\lceil rad(A_i)\rceil}$. Mittels des Morton Vergleichsoperators (in Algorithmus 2 mit $<$ und $>$ dargestellt) wird festgestellt, ob die beiden Punkte von p_{i-ck} und p_{i+ck} begrenzt werden, oder über diese Begrenzung hinaus reichen (Zeilen 5 und 10).
Ist dies nicht der Fall, wird der Suchraum mittels einer binären Suche in der jeweils betrachteten Suchrichtung angepasst bis ein Punkt gefunden wurde, der den *nearest neighbor Ball* weiter beschränkt (Zeilen 6-9 und 11-14). Der in der Binärsuche berechnete Punkt p_{i+2^I} oder p_{i-2^I} enthält an dieser Stelle die Binärrepräsentation 2^I, die ausgeschrieben für 2^2 zum Beispiel 10 ergibt und für 2^3 dementsprechend 100. Sinngemäß wird der Abstand von p_i zum gesuchten Punkt bei jeder Iteration verdoppelt, bis die Grenze von A_i erreicht wird oder die Abbruchbedingung in Zeile 8 oder 13 erfüllt wurde.
Diesen Teil des Algorithmus kann man sich wie ein Fenster der Länge $\mathcal{O}(k)$ vorstellen, dass über das sortierte Array der Punktwolke P geschoben wird.

Wurde in der *Sliding Window Phase* keine optimale Lösung gefunden, wird mit dem dritten Teil des Algorithmus fortgesetzt - der *Search Phase*. In Algorithmus 2 ist das die zweite Prozedur mit dem namen *CSEARCH*. Aufgabe der Prozedur ist es, die partielle Lösung aus A_i rekursiv zu verbessern, bis eine optimale Lösung gefunden wurde.
Die im zweiten Teil des Algorithmus berechneten neuen Werte für die Ober- und Untergrenze werden, soweit ihre Differenz eine Konstante ν nicht unterschreiten, zur Menge A_i hinzugefügt. In [4] wird $\nu = 4$ vorgeschlagen, dabei aber betont, dass der Wert plattformabhängig ist. Der genannte Wert hat sich im Rahmen dieser Arbeit aber ebenfalls als brauchbar erwiesen.
Aus den Werten für die neue Ober- und Untergrenze wird ein Mittelpunkt p_m berechnet und zusammen mit der Menge A_i zur erneuten Berechnung einer Nachbarschaft für Punkt p_i verwendet (Zeilen 20 - 21). Je nachdem, ob $p_i < p_m$ gilt wird die Prozedur CSEARCH erneut mit neuen Grenzen, links oder rechts von p_m liegend, aufgerufen (Zeilen 24 - 25 und Zeilen 27 - 28).

Algorithm 2 k-Nearest Neighbor Graph Contruction Algorithm

Require: Randomly shifted point set P of size n. Morton order compare operators: $<$,$>$. (COMPARE= $<$).

Ensure: A_i contains k nearest neighbors of p_i in P.

1: **procedure** CONSTRUCT(P, **int** k)
2: $P \leftarrow ParallelQSort(P, <)$
3: **parallel for all** p_i **in** P
4: $A_i \leftarrow nn^k(p_i, \{p_{i-ck}, ..., p_{i+ck}\})$
5: **if** $p_i^{\lceil rad(A_i) \rceil} < p_{i+ck}$ **then** $u \leftarrow i$
6: **else**
7: $I \leftarrow 1$
8: **while** $p_i^{\lceil rad(A_i) \rceil} < p_{i+2^I}$ **do** $++I$
9: $u \leftarrow min(i + 2^I, n)$
10: **if** $p_i^{-\lceil rad(A_i) \rceil} > p_{i-ck}$ **then** $l \leftarrow i$
11: **else**
12: $I \leftarrow 1$
13: **while** $p_i^{-\lceil rad(A_i) \rceil} > p_{i-2^I}$ **do** $++I$
14: $l \leftarrow max(i - 2^I, 1)$
15: **if** $l \neq u$ **then** CSEARCH(p_i,l,u)

16: **procedure** CSEARCH(**point** p_i, **int** l, **int** h)
17: **if** $(h - l) < \nu$ **then**
18: $A_i \leftarrow nn^k(p_i, A_i \cup \{p_l ... p_h\})$
19: **return**
20: $m \leftarrow (h + l)/2$
21: $A_i \leftarrow nn^k(p_i, A_i \cup p_m)$
22: **if** $dist(p_i, box(p_l, p_h)) \geq rad(A_i)$ **then return**
23: **if** $p_i < p_m$ **then**
24: CSEARCH($p_i, l, m - 1$)
25: **if** $p_m < p_i^{\lceil rad(A_i) \rceil}$ **then** CSEARCH($p_i, m + 1, h$)
26: **else**
27: CSEARCH($p_i, m + 1, h$)
28: **if** $p_i^{-\lceil rad(A_i) \rceil} < p_m$ **then** CSEARCH($p_i, l, m - 1$)

3.4 Parallelisierung

Aufgrund seines Aufbaus eignet sich der von *M. Connor* und *P. Kumar* in [4] beschriebene Algorithmus sehr gut für die parallelen Strukturen moderner Mehrkern Prozessoren. Die parallele Abarbeitung einer Punktwolke P findet in zwei Schritten statt.
Zunächst wird die Punktwolke, wie bereits in 3.3 beschrieben, mit Hilfe eines parallelen *Quick Sort* Algorithmus und des *Morton Order* Operators sortiert. Die Anzahl der Threads ist optimalerweise identisch mit der Anzahl der einzelnen Kerne des verwendeten Prozessors. Wurde das Array der Punktwolke einmal sortiert, finden keine weiteren schreibenden Operationen mehr darauf statt. Die Mengen A_i, welche für jeden Punkt berechnet werden, lassen sich sehr effizient bilden, indem nur die Indexwerte für jeden in der Menge befindlichen Punkt gespeichert werden. Auch die einzelnen Distanzen der Nachbarschaft müssen nicht separat gespeichert werden, wodurch der Speicherbedarf sehr übersichtlich bleibt. Da sich keine schreibenden Operationen auf dem Array der Punktwolke in die Quere kommen können, muss auch keine besondere Rücksicht auf eventuelle *Race Conditions* oder *Deadlocks* gelegt werden, was im Allgemeinen zu keinem Mehraufwand bei der Umsetzung des Algorithmus führt.

4 Matching

Dieses Kapitel setzt sich mit dem Matching Verfahren, welches im Rahmen dieser Arbeit entwickelt wurde, auseinander. Im Zuge dessen wird zunächst auf die Vorverarbeitung der aufgenommenen Graphen eingegangen, die die Handgesten für das Matching enthalten. Im Anschluss wird der Algorithmus selbst vorgestellt, seine Arbeitsweise ausgeleuchtet und es wird auf Besonderheiten eingegangen.

4.1 Transformation der Daten

Wie in Algorithmus 3 gezeigt, wird für jeden einzelnen Graphen eine Transformation durchgeführt, um ihn so auszurichten, dass er mit der aufgenommenen Punktwolke verglichen werden kann. Der Ablauf dieser Transformation ist für jeden Graphen gleich und besteht aus den folgenden drei Schritten:

1. Translation/Verschiebung
2. Rotation
3. Skalierung

Aufgrund der Hauptkomponentenanalyse (PCA), die für jeden Graphen durchgeführt wird, liegen die dafür nötigen Informationen bereits vor.
Die einzelnen Transformationen lassen sich mittels einfacher Matrixmultiplikationen durchführen. Im Zuge dessen müssen die Koordinaten des Graphen als erstes erweitert werden, um sie in eine homogene Form zu bringen. Die Koordinaten des Graphen sehen in Matrixnotation wie folgt aus:

$$G = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ g_{31} & g_{32} & \cdots & g_{3n} \end{bmatrix} \tag{12}$$

Erweitert man nun Matrix G, um homogene Koordinaten zu erhalten, ergibt sich folgende Matrix:

$$G_{hom} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ g_{31} & g_{32} & \cdots & g_{3n} \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{13}$$

Die einzelnen Transformationen werden durch Erweiterung der Einheitsmatrix I durchgeführt, die in der Diagonale mit Einsen gefüllt ist:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

Die Einheitsmatrix lässt sich für die Transformation des Graphen nutzen, da eine Multiplikation des Graphen mit der Einheitsmatrix zu keiner Veränderung führt ($I * G_{hom} = G_{hom}$). Änderungsoperationen lassen sich demnach mit der Einheitsmatrix I darstellen.
Auf die einzelnen Operationen wird im folgenden genauer eingegangen.

4.1.1 Translation

Die Punktwolkendaten des Graphen werden zum Koordinatenursprung hin verschoben. Genauer gesagt wird der Schwerpunkt der Wolke in den Ursprung verschoben. Dieser ist, dank der *PCA*, bereits als Vektor gegeben:

$$\mathbf{m} = \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} \tag{15}$$

Die sich daraus ergebende Translationsmatrix ergibt sich aus dem Vektor und der Einheitsmatrix:

$$T = \begin{bmatrix} 1 & 0 & 0 & m_1 \\ 0 & 1 & 0 & m_2 \\ 0 & 0 & 1 & m_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

Für eine Translation zum Ursprung muss aber die Inverse Translationsmatrix gebildet werden

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -m_1 \\ 0 & 1 & 0 & -m_2 \\ 0 & 0 & 1 & -m_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

und anschließend mit der homogenisierten Matrix des Graphen multipliziert werden:

$$G_{trans} = T^{-1} * G_{hom} \tag{18}$$

Abbildung 9 zeigt die Translation am Beispiel einer Punktwolke. Im linken Bild befindet sich die Punktwolke an ihrer ursprünglichen Position, während ihr Schwerpunkt im rechten Bild in den Koordinatenursprung verschoben wurde.

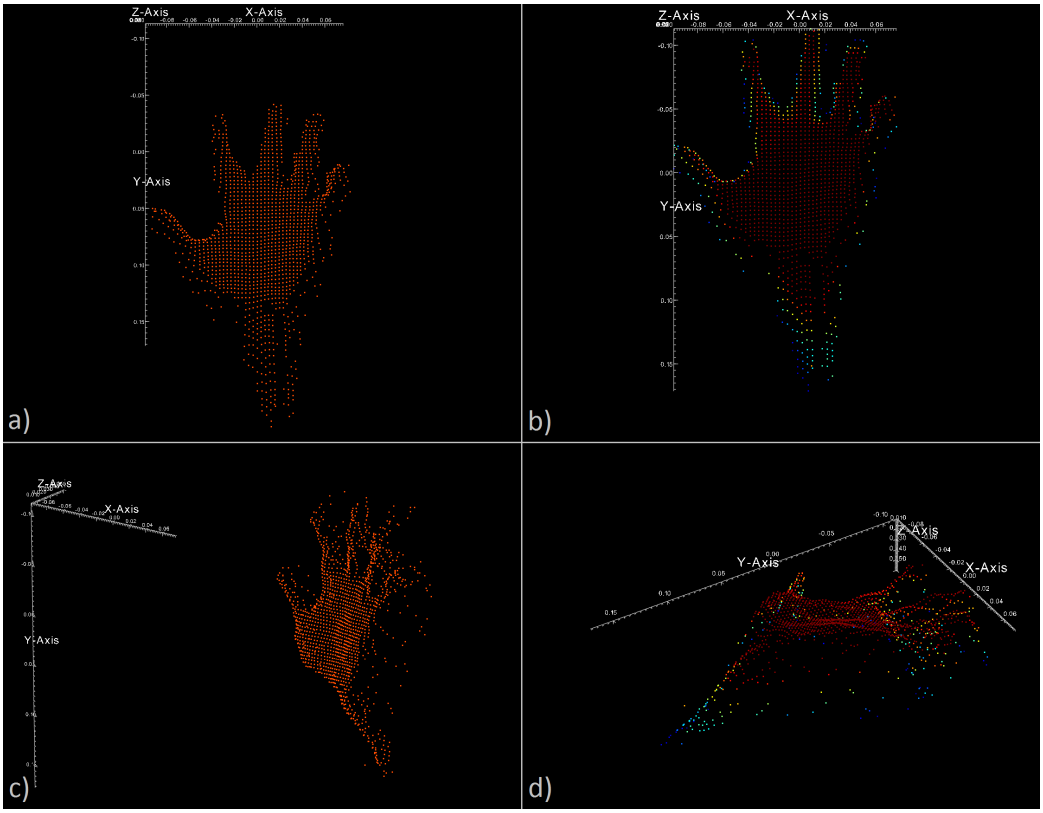

Abbildung 9: Translation einer Punktwolke: (a) zeigt die Punktwolke in ihrer ursprünglichen Position und (b) nachdem sie über ihren Schwerpunkt in den Ursprung verschoben wurde. (c) und (d) zeigen die gleiche Situation aus seitlichen Ansichten, um die Verschiebung in z-Richtung zu verdeutlichen.

4.1.2 Rotation

Wurde die Punktwolke des Graphen in den Ursprung verschoben kann sie im nächsten Schritt um ihren Schwerpunkt rotiert werden. Wie auch bei der Translation lässt sich das mit Hilfe der Einheitsmatrix lösen. Hierfür werden

die drei Hauptkomponenten benötigt, die in Matrixnotation folgendermaßen darstellbar sind:

$$C = \begin{bmatrix} c_{11} & c_{21} & c_{31} \\ c_{12} & c_{22} & c_{32} \\ c_{13} & c_{23} & c_{33} \end{bmatrix} \tag{19}$$

Die Rotationsmatrix ergibt sich aus den Hauptkomponenten aus C und der Einheitsmatrix I:

$$R = \begin{bmatrix} c_{11} & c_{21} & c_{31} & 0 \\ c_{12} & c_{22} & c_{32} & 0 \\ c_{13} & c_{23} & c_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{20}$$

deren Inverse für die Transformation benötigt wird:

$$R^{-1} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & 0 \\ c_{21} & c_{22} & c_{23} & 0 \\ c_{31} & c_{32} & c_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{21}$$

Auch hier wird anschließend mit $G_{rot} = R^{-1} * G_{trans}$ wieder eine Matrixmultiplikation durchgeführt. Das Ergebnis solch einer Rotation ist in Abbildung 10 dargestellt.

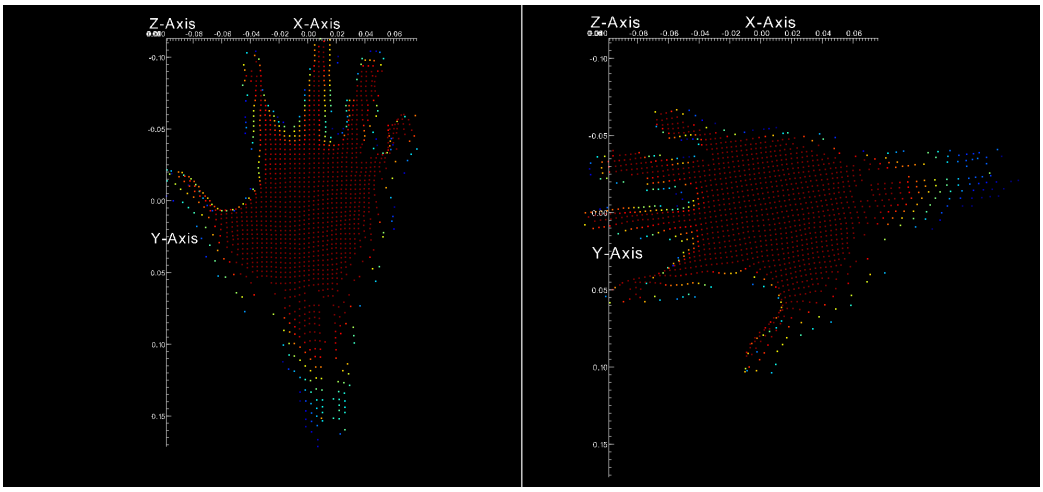

Abbildung 10: Rotation einer in den Ursprung verschobenen Punktwolke: Links befindet sich die Wolke in ihrer ursprünglichen Ausrichtung, während sie rechts über ihre Hauptkomponenten rotiert wurde.

4.1.3 Skalierung

Wurde der Graph, wie in den Abschnitten 4.1.1 und 4.1.2 vorgestellt, verschoben und rotiert, muss als letztes noch sichergestellt werden, dass er die richtige Größe aufweist. Es muss also eine geeignete Skalierung erfolgen, die genau das ermöglicht. Ein einfacher Weg ist es, den Graphen mittels der PCA zu skalieren, da die Eigenwerte im Grunde eine Gewichtung der Hauptkomponenten darstellen. Der Eigenvektor

$$\boldsymbol{\lambda} = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} \tag{22}$$

bildet zusammen mit der Einheitsmatrix I die Skalierungsmatrix:

$$S = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{23}$$

Soll eine Punktwolke auf die Größe einer zweiten Punktwolke skaliert werden, wird zusätzlich zur gezeigten Skalierung noch die Inverse benötigt:

$$S^{-1} = \begin{bmatrix} \frac{1}{\lambda_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\lambda_2} & 0 & 0 \\ 0 & 0 & \frac{1}{\lambda_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{24}$$

Soll der Graph anschließend auf die Größe der Punktwolke gebracht werden, müssen insgesamt zwei Multiplikationen durchgeführt werden. Als erstes muss der Graph verkleinert werden. Das geschieht mit der inversen Skalierungsmatrix mit den Werten aus dem Eigenvektor der PCA des Graphen (S_G^{-1}). Anschließend wird die Skalierungsmatrix aus dem Eigenvektor der PCA der Punktwolke gebildet, um die Vergrößerung zu ermöglichen (S_P). Die gesamte Transformation ergibt sich dann durch:

$$G_{ska} = S_P * S_G^{-1} * G_{rot}. \tag{25}$$

Die Abbildungen 11 und 12 veranschaulichen die inverse Skalierung und die darauf folgende nicht-inverse Skalierung anhand einer beispielhaften Punktwolke.

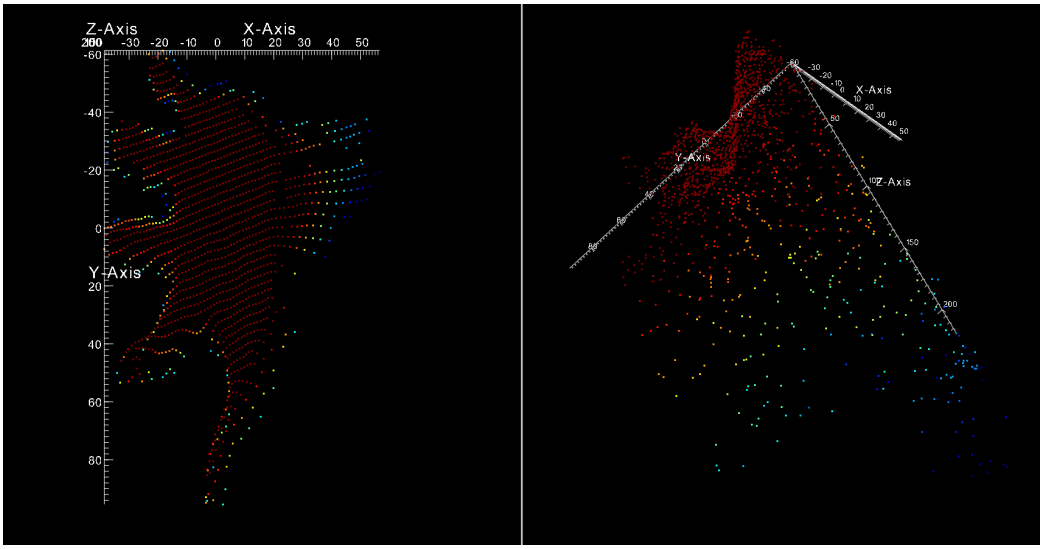

Abbildung 11: Inverse Skalierung einer Punktwolke: Zu erkennen ist die aus Abbildung 10 gezeigte Punktwolke, die mit Hilfe der inversen Skalierungsmatrix S^{-1} transformiert wurde. Während sich in der Draufsicht (links) noch erkennen lässt, um was für eine Handgeste es sich handelt, ist die seitliche Ansicht (rechts) sehr verzerrt. Auffällig ist auch die Größe der Punktwolke, die um ein vielfaches zugenommen hat (zu erkennen an den Koordinatenachsen).

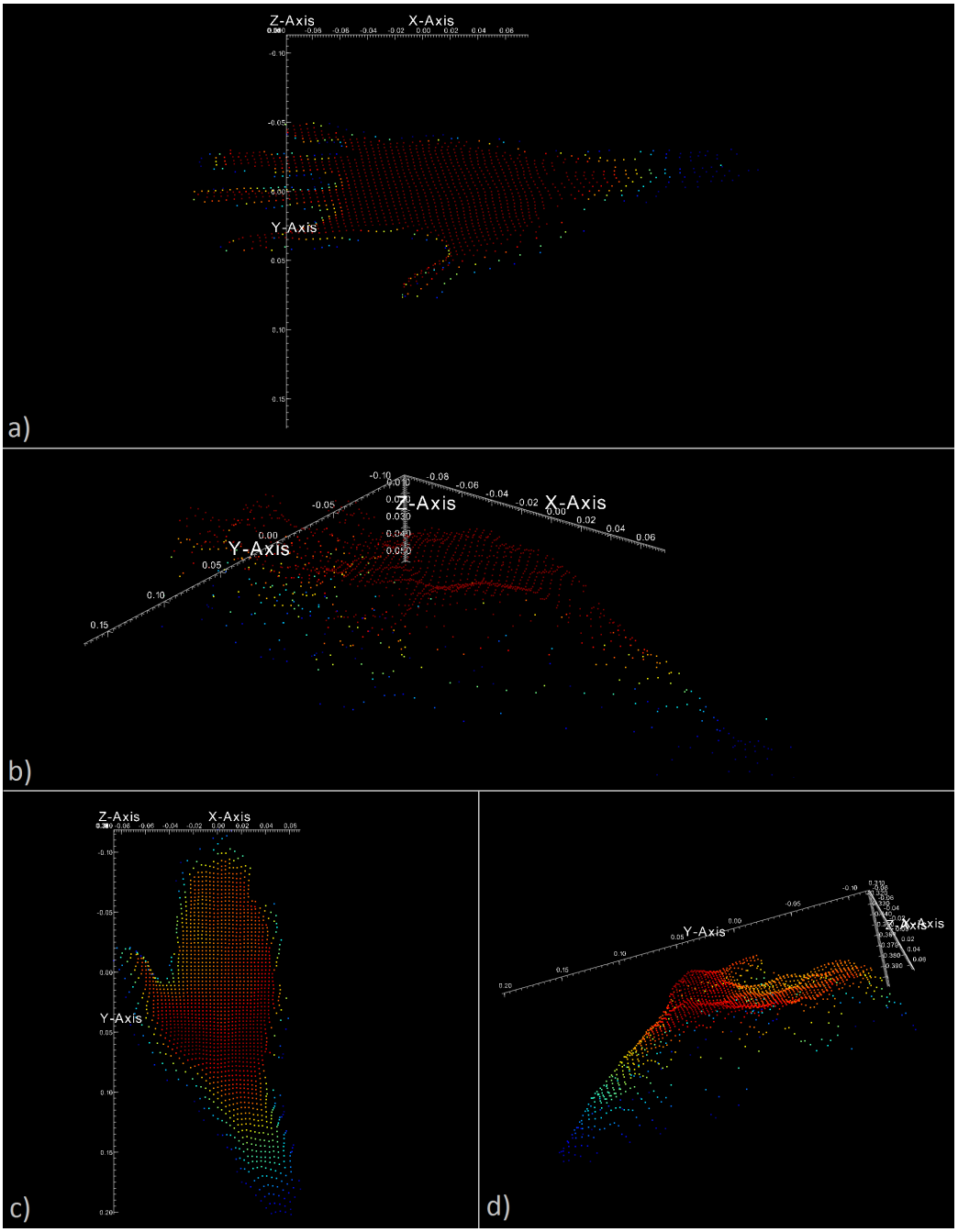

Abbildung 12: Skalierung einer Punktwolke: Die zuvor invers skalierte Punktwolke ist nun mit Hilfe der Skalierungsmatrix S auf die Größe der zu vergleichenden Punktwolke skaliert worden. (a) und (b) zeigen zwei verschiedene Ansichten der transformierten Wolke. Zum Vergleich ist in (c) und (d) die zu vergleichende (nicht transformierte) Punktwolke abgebildet. Man sieht deutlich, dass die Handgeste aus dem Graphen (a und b) an die Maße der Punktwolke (c und d) skaliert wurde.

4.1.4 Ausrichtung des Graphen

Die Hauptkomponentenanalyse hat einen gravierenden Nachteil, der bei der Rotation des Graphen in Richtung der Punktwolke sichtbar wird. Die Ausrichtung der Hauptkomponenten ist nicht eindeutig, wodurch die PCA nicht bestimmen kann, ob ein Graph zum Beispiel auf dem Kopf steht. Jede einzelne Hauptkomponente könnte auch um 180 Grad rotiert sein. Im Ergebnis der PCA lässt sich das nicht ablesen. Um dennoch eine genaue Ausrichtung des Graphen zur Punktwolke hin zu erhalten lässt sich das Vorzeichen der jeweiligen Dimension/Hauptkomponente berechnen. Diese ergeben sich für eine Punktwolke, die n Punkte enthält, folgendermaßen:

$$s_x = \sum_{i=1}^{N} x_i^3, \tag{26}$$

$$s_y = \sum_{i=1}^{N} y_i^3, \tag{27}$$

$$s_z = \sum_{i=1}^{N} z_i^3. \tag{28}$$

Sollte das Vorzeichen von s_x, s_y oder s_z negativ sein muss das Vorzeichen für jede Koordinate des Graphen in der jeweiligen Dimension umgedreht werden. Die gleiche Berechnung muss auch für die zu vergleichende Punktwolke geschehen, um anschließend nur noch positive Werte für die in den Gleichungen 26, 27 und 28 erhaltenen Werte zu bekommen.

4.2 Algorithmus

Der *Graphh Matching* Algorithmus besteht aus insgesamt 2 Phasen:

1. Preprocessing
2. Graph Matching

In der *Preprocessing*-Phase, dargestellt in Algorithmus 3, werden zunächst alle verfügbaren Graphen geladen, die die einzelnen Handgesten enthalten (Zeile 2). Anschließend wird parallel für alle Graphen eine Hauptkomponentenanalyse durchgeführt und dann nacheinander die in den Abschnitten 4.1.1

und 4.1.2 vorgestellten Transformationen durchgeführt, um die Graphen in den Ursprung zu verschieben und zu rotieren (Zeilen 4-6). Zuletzt wird die Ausrichtung des Graphen ermittelt und gegebenenfalls korrigiert (Zeilen 7-9).

Algorithm 3 Graph Preprocessing Algorithm

Require: List of unchanged Graphs GL of size n.

Ensure: List of transformed Graphs GL and the list of their corresponding PCAs $GraphPCAList$.

```
1:  procedure PREPROCESS(GraphList GL)
2:      GraphPCAList ← {graphPCA_1, ..., graphPCA_n}
3:      parallel for all G_i in GL do
4:          graphPCA_i ← computePCA(G_i)
5:          moveToOrigin(G_i, meanValues_graphPCA_i)
6:          rotate(G_i, principalComponents_graphPCA_i)
7:          s_i ← {s_i^x, s_i^y, s_i^z}
8:          s_i ← calculateSigns(G_i)
9:          switchGraphSigns(G_i, s_i)
10:     end parallel for
11:     return GraphPCAList, GL
```

Die zweite Phase bildet den Kern des Matching Verfahrens und ist in Algorithmus 4 abgebildet. Sie besteht aus einer Hauptprozedur, die für eine aufgenommene Punktwolke über alle Graphen iteriert und den besten Kandidaten heraussucht und aus einer Nebenprozedur, die die Berechnung für jeden Graphen durchführt.
Im Folgenden wird zunächst auf die Hauptprozedur, in Algorithmus 4 als FINDWINNER bezeichnet, eingegangen, um den allgemeinen Ablauf vorzustellen. Die Berechnung der Distanz zwischen Graph und Punktwolke, die in Prozedur DISTGRAPH durchgeführt wird, wird dann am Ende dieses Abschnitts detailliert beschrieben.
Im ersten Schritt wird die Liste der Graphen mit Hilfe der in Algorithmus 3 vorgestellten Prozedur vor verarbeitet (Zeile 13) und es wird die PCA der aktuellen Punktwolke P berechnet (Zeile 14). Wie auch für die Graphen wird eine Transformation auf den Daten der Punktwolke durchgeführt, die aus einer Verschiebung und einer Rotation besteht. Des Weiteren werden auch die Vorzeichen der Punktwolke berechnet und gegebenenfalls korrigiert (Zeilen 15 - 19). Danach sind alle Graphen sowie die Punktwolke transformiert und sollten im optimalen Fall die gleiche Ausrichtung haben. Bevor die Graphen mit der Punktwolke verglichen werden können, müssen sie noch in einer

letzten Transformation auf die richtigen Maße skaliert werden. Auf diese Weise wird das Matching Verfahren gegenüber variierenden Aufnahmedistanzen oder Handgrößen robuster. Das Skalieren findet, anders als die Translation und die Rotation, nur auf den Graphen statt, da die betrachtete Punktwolke ansonsten für jeden Graphen erneut skaliert werden müsste (Zeile 22).
Pro Graph wird im letzten Schritt des Verfahrens die Prozedur DISTGRAPH aufgerufen. Aus dem Ergebnis kann dann ein durchschnittlicher Fehler berechnet werden, der als Maß für die Ähnlichkeit von Punktwolke und Graph verstanden werden kann. Der Graph, der den geringsten Fehler aufweist, sollte demnach auch die größte Ähnlichkeit zur Punktwolke haben und wird als Gewinner betrachtet.

Die Prozedur DISTGRAPH, abgebildet in Algorithmus 4, ist für die Berechnung der Distanz zwischen einem einzelnen Graphen und der Punktwolke zuständig. Genauer gesagt wird berechnet, wie hoch die Distanz jedes Punktes in der Punktwolke P zu seinem nächstgelegenen Nachbarn im zu vergleichenden Graphen ist.
Die Prozedur läuft parallel für jeden Punkt in der Punktwolke ab und iteriert jeweils über den gesamten Graphen (Zeile 3-5). Für jeden Punkt im Graphen wird seine Distanz zum betrachteten Punkt in der Punktwolke berechnet (Zeile 6). Dabei wird die jeweils geringste Distanz gespeichert, da an dieser Stelle der nächstgelegene Nachbar gefunden wurde (7-9). Als Ergebnis wird dann die Liste der minimalen Distanzen zurückgegeben (Zeile 11).

Algorithm 4 Gesture Recognition Algorithm

Require: List of Graphs GL of size n and input point cloud P of size m.
Ensure: Winner Graph G with minimal average error.

1: **procedure** DISTGRAPH(**PointCloud** P, **Graph** G)
2: $Dist \leftarrow \{dist_1, ..., dist_m\}$
3: **parallel for all** p_i **in** P **do**
4: $minDist \leftarrow MAX\_DOUBLE$
5: **for** $j = 0$ **to** m **do**
6: $tempDist = squaredDist(p_i, G.point_i)$
7: **if** $tempDist < minDist$ **then**
8: $minDist = tempDist$
9: $dist_j = minDist$
10: **end parallel for**
11: **return** $Dist$

12: **procedure** FINDWINNER(**GraphList** GL, **PointCloud** P)
13: $GraphPCAList, GL =$ PREPROCESS(GL)
14: $cloudPCA \leftarrow$ computePCA(P)
15: moveToOrigin(P,$meanValues_{cloudPCA}$)
16: rotate(P,$principalComponents_{cloudPCA}$)
17: $s_P \leftarrow \{s_P^x, s_P^y, s_P^z\}$
18: $s_P \leftarrow$ calculateSigns(P)
19: switchSigns(P,s_P)
20: $MinDist \leftarrow \{minDist_1, ..., minDist_n\}$
21: **for all** G_i **in** GL **do**
22: scale(G_i,$graphPCA_i$,$cloudPCA$)
23: $Dist \leftarrow$ DISTGRAPH(P, G_i)
24: $minDist_i \leftarrow$ average($Dist$)
25: $winnerGraph =$ min($MinDist$)
26: **return** $winnerGraph$

5 Implementierung

Im Folgenden wird kurz auf die verwendete Programmiersprache, Entwicklungsumgebung sowie die Libraries eingegangen, die für die Umsetzung der vorgestellten Algorithmen verwendet wurden.

5.1 Libraries und Entwicklungsumgebung

Der gesamte Programmcode ist in der Programmiersprache *C++* umgesetzt. Für die Entwicklung wurde *Visual Studion 2008 Professional* verwendet, wodurch auch der *C++*-Compiler auf diese Version beschränkt ist. Der Grund hierfür war das Entwicklungsframework *Adaf*, das bis dato nur mit dieser Version kompatibel war und andernfalls zu unvorhersehbarem Verhalten führen konnte.
Des weiteren wurden Funktionen der Libraries *PRAGMA*, *Boost* und *Adaf* verwendet.
PRAGMA (**P**attern **R**ecognition **A**nd **G**raph **M**atching **A**lgortihms) ist eine am Institut für Neuroinformatik entwickelte Library, die eine Vielzahl an Algorithmen und Funktionen für die Objekt- sowie Personenerkennung bietet. Verwendet wurde sie im Speziellen für die in 2.4 beschriebene Hauptkomponentenanalyse und die Transformation der Punktwolken und Graphen, auf die in 4.1 eingegangen wurde.
Da es sich bei *Adaf* um mehr als eine Library handelt, wird diese in Abschnitt 5.2 detailliert behandelt. Verwendet wurden aber hauptsächlich die Funktionen, die für den Bau der *Adaf*-Module notwendig sind, sowie Datentypen für die einzelnen Datensätze wie zum Beispiel Punktwolken und einzelne Punkte.
Die Library *Boost* hingegen kam zum Einsatz, da sie von *Adaf* und *PRAGMA* benötigt wird.
Eine weitere Library, die für die Entwicklung in Frage kam, nach intensiverem Testen jedoch verworfen werden musste, war *PCL* - auch als **P**oint **C**loud **L**ibrary bekannt [5]. Trotz vielversprechender Funktionalitäten war sie leider nicht mit der verwendeten Version des *C++* Standards kompatibel und führte zu nicht lösbaren Problemen während der Entwicklung.

5.2 Adaf

Bei Adaf (*Advanced Development & Analysis*) handelt es sich um ein Entwicklungsframework, dass die Aufzeichnung und die Verarbeitung von zwei- und dreidimensionalen Sensordaten erlaubt. Zusammen mit dem Adaf SDK für die Programmiersprache *C++* bildet es ein Grundgerüst, mit dem die Entwicklung von Bildverarbeitungssystemen ermöglicht wird.

5.2.1 Adaf Framework

Die Verarbeitung von Daten findet auf Basis von Modulen statt. Jedes Modul bildet eine in sich geschlossene Einheit, welche über sogenannte Pins mit anderen Modulen kommunizieren kann. Jedem Pin ist ein fester Datentyp zugeordnet, der sicherstellt, dass die passenden Daten am Modul ankommen, oder dieses wieder verlassen.
Abbildung 13 gibt eine Übersicht über die Oberfläche des Adaf Frameworks. In der linken oberen Ecke sind die einzelnen Module des geladenen Beispielprojekts zu erkennen. Diese werden auch als Plugins bezeichnet. Die einzelnen Plugins sind über ihre Pins miteinander verbunden, wobei sich Input- und Output-Pins voneinander unterscheiden.
Die Aufnahme der Sensordaten findet auch über ein solches Plugin statt, welches zudem kameraspezifisch ist. Je nach Kameratyp stellt es verschiedene Output-Pins zur Verfügung, welche die jeweiligen Daten an weitere Plugins weitergeben können (zu erkennen an dem linken blauen Rechteck in der linken oberen Ecke).
Für die Visualisierung von Daten liegen dem Framework ebenfalls einige Plugins bei. Im Rahmen dieser Arbeit findet vor allem das Plugin für die Darstellung dreidimensionaler Daten Anwendung. Als Input erhält es Punktwolken-Daten und stellt diese über eine visuelle Ausgabe dar. Zu erkennen ist dies in der Mitte von Abbildung 13. Die dreidimensionalen Daten lassen sich aus allen Perspektiven frei betrachten und geben dem Entwickler somit die Möglichkeit einer genauen Analyse.

5.2.2 Adaf SDK

Zusätzlich zum Framework bietet Adaf auch ein SDK für die Programmiersprache *C++* an. Es beinhaltet eine größere Menge an Datentypen und Klassen, die für die Entwicklung mit dem Framework nötig sind oder zumindest die Arbeit damit erleichtern. Eine genaue Übersicht über die API lässt sich der Dokumentation entnehmen, welche dem Adaf Framework beiliegt.

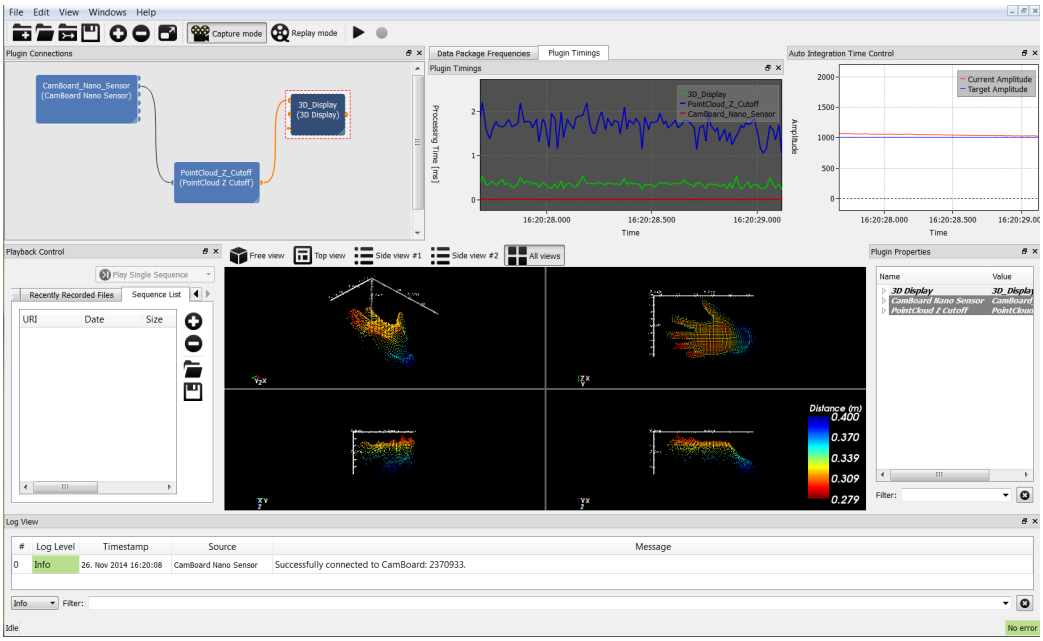

Abbildung 13: Das Adaf Framework.

6 Testläufe

Das Matching Verfahren wurde im Rahmen dieser Arbeit ausführlichen Testläufen unterzogen, um die Qualität der Erkennung von Handgesten evaluieren zu können. Dieses Kapitel befasst sich mit den Testläufen und analysiert den in Kapitel 4 vorgestellten Algorithmus anhand der gewonnenen Resultate.
Im Zuge dessen wird zunächst auf den Versuchsaufbau eingegangen. Dieser umfasst die Einstellung des *Adaf* Frameworks sowie die Konfiguration der benötigten Plugins und der verwendeten TOF-Kamera. Anschließend wird die Erstellung der einzelnen Graphen für die Handgesten, wie es in Kapitel 3 beschrieben wurde, vorgestellt. Dabei liegt der Fokus nicht nur auf den Graphen selbst, sondern auch auf der Laufzeit während der Aufnahme.
Ebenfalls wird auf die einzelnen Handgesten eingegangen, die für die Versuche zum Einsatz kommen. Es werden Gründe für die Wahl der jeweiligen Handgesten genannt und mögliche Nachteile, die beim Matching auftreten könnten besprochen. Der Ablauf des Matchings selbst wird danach vorgestellt, um ein Verständnis über die Arbeitsweise zu geben.
Nachdem der gesamte Aufbau und Ablauf beschrieben wurde, werden im letzten Abschnitt die Testreihen vorgestellt und detailliert ausgewertet.

6.1 Versuchsaufbau

Um sicherzustellen, dass zwischen der Aufnahme der Graphen und dem Matching der Handgesten keine qualitativen Unterschiede bezüglich Beleuchtung und Position entstehen, wurde jeweils der gleiche Versuchsaufbau für beide Phasen der Testläufe gewählt.
Zum Einsatz kam ein Workstation Laptop mit *Intel Core i7-2820QM* mit maximal 3.4 GHz und 16 Gigabyte RAM. Sämtliche Aufnahmen fanden in einer sitzenden Position an einem Schreibtisch statt, wobei die TOF-Kamera in einer Höhe von 40 Zentimetern zur Tischplatte befestigt wurde.
Einzig in der Wahl der geladenen Plugins für das *Adaf*-Framework unterscheiden sich die beiden Phasen. Abbildung 14 zeigt eine Übersicht über alle verwendeten Plugins. Die in der ersten Phase verwendeten Plugins sind grün markiert, während das in der zweiten Phase zum Einsatz kommende Plugin rot markiert ist. Alle gelb markierten Plugins werden in beiden Testphasen verwendet.

6.2 Generierung der Graphen

Die richtige Wahl der Parameter bei der Aufnahme der Graphen beeinflusst maßgeblich deren Qualität und dadurch vor allem auch die Qualität des

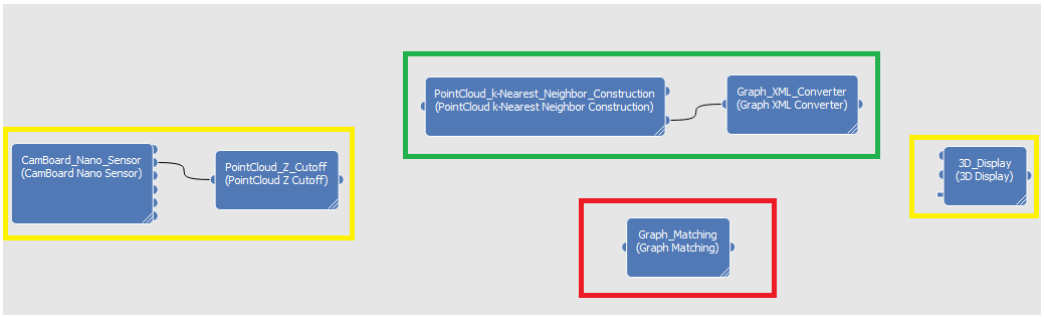

Abbildung 14: Übersicht über alle verwendeten *Adaf*-Plugins.

Matching Verfahrens. Für die Generierung der Graphen wurden zwei *Adaf*-Plugins entwickelt, die bereits in Abbildung 14 gezeigt wurden (rot markiert). Die einzelnen Aufgaben der Plugins *PointCloud k-Nearest Neighbor Construction* und *Graph XML Converter* lassen sich bereits an deren Namen ablesen. Während ersteres Plugin für die eigentliche Generierung der Graphen zuständig ist, speichert das zweite Plugin die fertigen Graphen im XML Format ab.
Die Aufnahme läuft dabei in drei Schritten ab. Zuerst werden mit Hilfe des *PointCloud Z Cutoff* Plugins alle Punkte der aufgenommenen Punktwolke entfernt, die sich außerhalb eines definierten Distanzintervalls befinden. Dieses Intervall ist frei konfigurierbar, jedoch haben sich die in Abbildung 16 gezeigten Parameter für *Z Cutoff Threshold* und *Z Cutoff Threshold Max* als sehr solide erwiesen und kamen deswegen während allen Versuchsphasen zur Anwendung. Dieses Vorgehen hat zwei entscheidende Vorteile. Einerseits bietet es eine einfache Möglichkeit störende Elemente aus der Punktwolke zu entfernen, da man davon ausgehen kann, dass weit entfernte Punkte wahrscheinlich nicht Teil einer Hand sind. Auch das Entfernen zu naheliegender Punkte ist sinnvoll, da die TOF-Kamera zu nahe Objekte nur noch verzerrt darstellt. Andererseits ergibt sich ein enormer Geschwindigkeitszuwachs, da die aufgenommene Punktwolke um ein Vielfaches verkleinert wird. Somit muss anschließend nur noch ein Bruchteil der Punkte verarbeitet werden. Zum Vergleich: Die TOF-Kamera hat eine Auflösung von $165x120$ Punkten, was einer Gesamtmenge von 19.800 Punkten pro Punktwolke entspricht. Werden zuvor alle Punkte entfernt, die sich nicht im vorgegebenen Intervall befinden, verkleinert man die Punktwolke auf durchschnittlich 5.000 - 10.000 Punkte.
Im zweiten Schritt der Aufnahme wird aus der verkleinerten Punktwolke ein Graph erstellt. Das Plugin *PointCloud k-Nearest Neighbor Constructi-*

on bietet zwei Parameter, die einen direkten Einfluss auf den resultierenden Graphen haben:

1. *Neighborhood Size*
2. *Neighborhood Distance Threshold*

Der erste Parameter gibt an, wie groß die Nachbarschaften für jeden einzelnen Punkt in der Punktwolke ausfallen sollen. Die Wahl beschränkt sich dabei auf Nachbarschaften der Größen $2, 4, 8, 16$ und 32, wobei die beiden letzten Werte für die Versuche nicht zur Anwendung kommen. Der zweite Parameter bestimmt, wie groß der maximale Abstand zwischen dem betrachteten Punkt und dem am weitesten entfernten Nachbarn sein darf, um in den Graphen aufgenommen zu werden. Je nach Wahl der beiden Parameter variiert die Dichte der resultierenden Graphen. Einen Vergleich zwischen drei unterschiedlichen Nachbarschaftsgrößen zeigt Abbildung 15. Links wurde der Graph mit einer Nachbarschaft von 2 Punkten erstellt, in der Mitte mit einer Nachbarschaft von 4 Punkten und rechts mit einer Nachbarschaft von 8 Punkten.

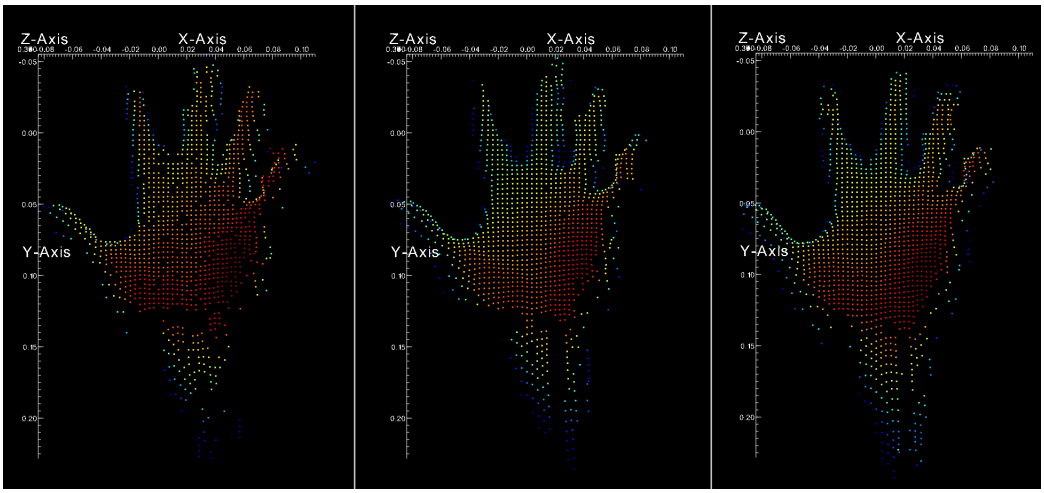

Abbildung 15: Punktwolken einer Hand mit drei unterschiedlichen Nachbarschaftsgrößen: Der Graph der links abgebildeten Punktwolke besteht aus 1241 Punkten und 940 Kanten, der Graph der in der Mitte abgebildeten Punktwolke besteht aus 1333 Punkten und 2062 Kanten und der Graph der rechts abgebildeten Punktwolke besteht aus 1425 Punkten und 4383 Kanten.

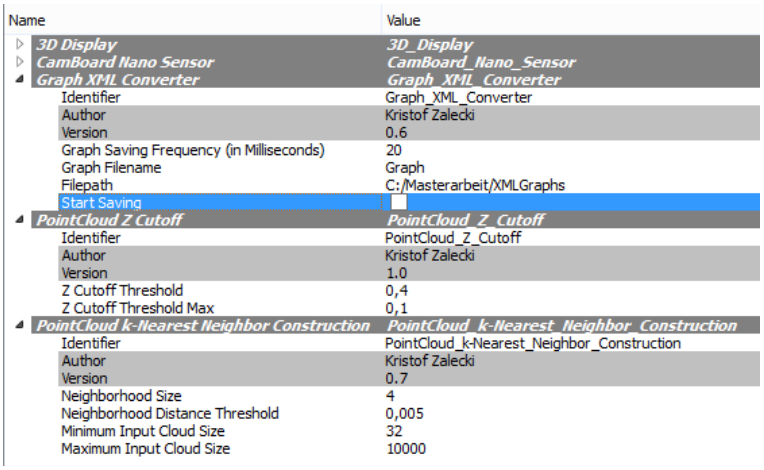

Abbildung 16: Übersicht über die *Adaf*-Plugins, die für die Erstellung von Graphen benötigt werden.

6.3 Die Handgesten

Um aussagekräftige Ergebnisse zu ermöglichen, wurden acht repräsentative Handgesten aufgenommen, welche aus teilweise sehr unterschiedlichen Gründen gewählt wurden. Hierbei wurde vornehmlich das Ziel verfolgt, eine möglichst große Bandbreite an Eigenschaften mit einer relativ überschaubaren Menge an Daten abzudecken.
Nach intensiver Suche stellten sich die folgenden Handgesten als gute Kandidaten für die Testreihen heraus:

1. *Hand Vorne*
2. *Hand Hinten*
3. *Hand Vorne Finger Zusammen*
4. *Daumen*
5. *Greifen*
6. *Faust*
7. *OK*
8. *Victory*

Nachfolgend werden diese Handgesten einzeln vorgestellt und auf deren Eigenschaften und mögliche Vor- und Nachteile im Bezug auf die Handgestenerkennung eingegangen.

Die Handgeste *Hand Vorne* bildet - wie der Name sagt - eine flache Hand von vorne ab. Die Finger sind leicht gespreizt. Der Vorteil ist, dass es sich um eine recht einfache Handgeste handelt, die zudem auch eine große Fläche hat und somit eine dementsprechend ausgeprägte Menge an Punkten aufweist.
Zum direkten Vergleich steht die Handgeste *Hand Hinten*. Diese stellt, wie auch *Hand Vorne*, eine flache Hand mit leicht gespreizten Fingern dar, jedoch von der Rückseite. Interessant an dieser Geste ist die Frage, wie gut die Erkennung in Verbindung mit der ersten Handgeste umgehen kann. Betrachtet man die beiden Handgesten genauer, lässt sich aus der Frontalansicht kein großer Unterschied feststellen. Wenn man in Betracht zieht, dass die PCA eine linke von einer rechten Hand nicht unterscheiden kann und die Ausrichtung der Hand unter Umständen auch keine Rollen spielt, könnte es an dieser Stelle also bereits zu einer Verwechslung kommen, die es zu untersuchen gilt.
Abbildung 17 stellt die beiden Handgesten - dargestellt unter (a) und (b) - in einen direkten Vergleich. Auffällige Unterschiede werden erst sichtbar, wenn man die seitlichen Ansichten betrachtet. Während bei der von vorne betrachteten Hand eine Kuhle in der Handfläche erkennbar ist, weist die Hand von hinten betrachtet eine Wölbung nach außen auf, die den entscheidenden Unterschied bei der Erkennung liefern könnte.
Eine weitere Variation der flachen Hand ist zudem mit der Handgeste *Hand Vorne Finger Zusammen* gegeben. Wie auch bei der ersten Handgeste wird eine flache Hand abgebildet, nun jedoch mit aneinanderliegenden Fingern, sodass die Handfläche zusammen mit den Fingern eine geschlossene Oberfläche bildet. Vergleicht man diese Handgeste (in Abbildung 17 unter (c) dargestellt) mit der ersten, ergibt sich rein visuell ein größerer Unterschied, als zuvor zwischen den beiden Händen unter (a) und (b). Nachteilig kann sich an dieser Stelle aber die Skalierung der Graphen (beschrieben in Kapitel 4 Abschnitt 4.1) auswirken, die den Unterschied der Proportionen womöglich enorm minimiert. Somit stehen die drei vorgestellten Handgesten in einer direkten Konkurrenz zueinander und könnten zu ähnlichen Vergleichswerten oder gar Fehlerkennungen führen.

Die Handgeste *Daumen* hebt sich von den drei bisher vorgestellten Handgesten sehr ab. Wie in Abbildung 18 unter (a) zu erkennen ist, handelt es sich um eine seitlich zur Faust geballte Hand mit hoch gestrecktem Daumen. Im Gegensatz zur flachen Hand ist in der Frontalansicht nur eine recht kleine Fläche zu erkennen. In der Seitenansicht zeigt sich die größte Oberfläche, die

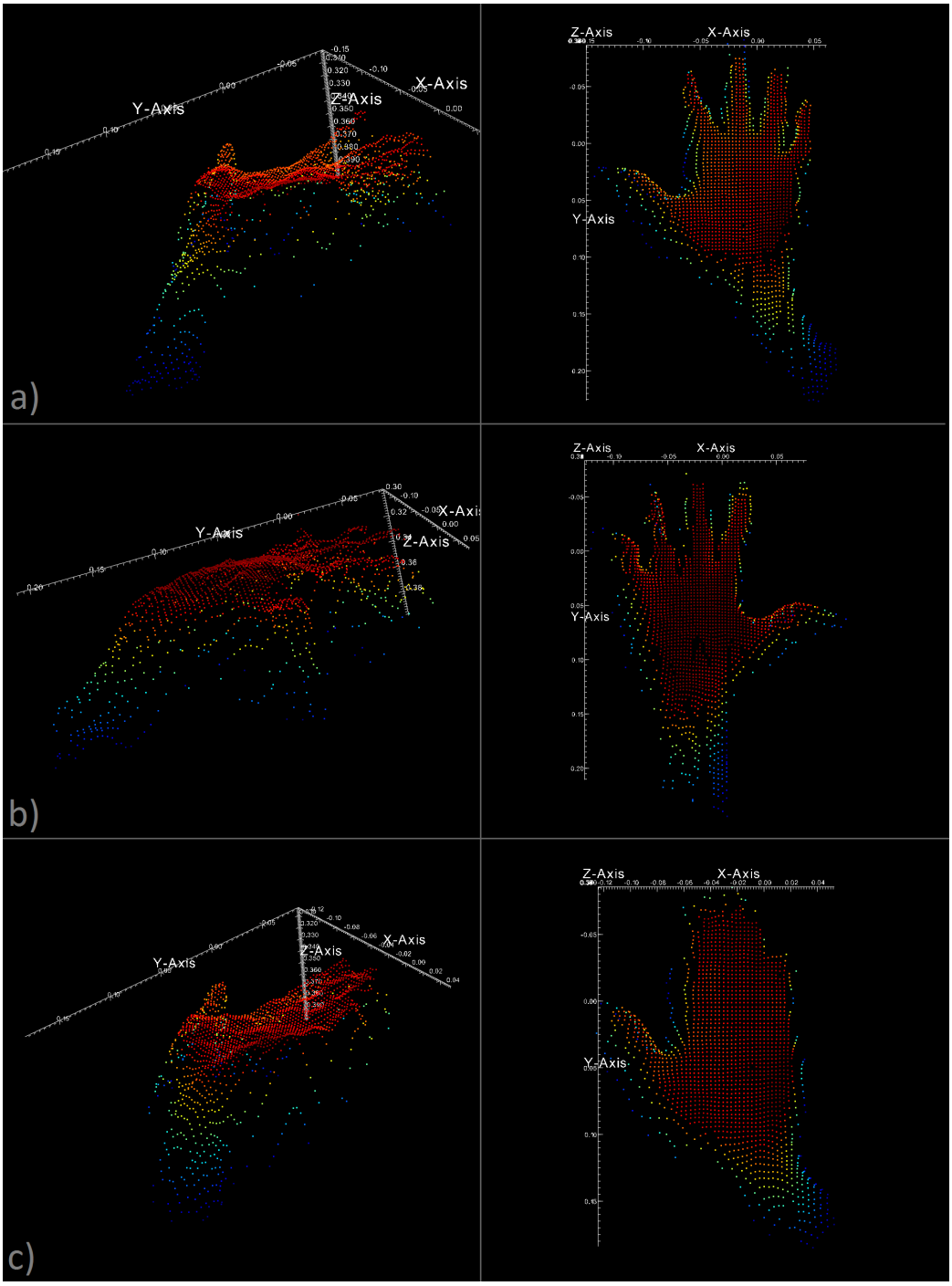

Abbildung 17: Übersicht über die Handgesten *Hand Vorne*, *Hand Hinten* und *Hand Vorne Finger Zusammen*.

aber nur aus wenigen Punkten besteht. Diese Handgeste wurde in die Testmenge aufgenommen, da sie allgemein bekannt ist und sich darüber hinaus sehr von den anderen abhebt. Es lässt sich somit testen, wie gut verschiedene Formen und Ausrichtungen im Allgemeinen erkannt werden können. Des weiteren kann hier auch unter Umständen geprüft werden, wie gut die Transformation mit Hilfe der PCA auf diese ‚von vorne betrachtet sehr gleichmäßige Form, angewendet wird, um sie zu rotieren und zu skalieren. Denkbar wäre zum Beispiel, dass die Lage des Daumens bereits ausreichen könnte, um die Transformation ausreichend zu beeinflussen oder die recht große Fläche an den Seiten des Daumens fälschlicherweise auf die Fläche einer anderen Handgeste passen könnte.
Die Handgeste *Greifen* (Abbildung 18 unter (c)) weist eine nicht unwesentliche Ähnlichkeit zu den Gesten *Hand Vorne* und *Hand Hinten* auf. Im Gegensatz zu deren gespreizten Fingern, sind diese aber gekrümmt und erinnern an eine greifende Hand, wie zum Beispiel bei einer Hand, die einen faustgroßen Ball hält. Besonders in der seitlichen Ansicht ist die Krümmung der Finger erkennbar, die sich deutlich vom Rest der Hand abheben. Die Kuhle in der Mitte der Handfläche ist deutlich ausgeprägter als bei *Hand Vorne* und die Länge der Finger wirkt aus der Frontalansicht leicht verkürzt. Gewählt wurde diese Handgeste unter anderem, weil geprüft werden soll, ob alleine die Lage der Finger ausreicht, um eine eindeutige Erkennung zu ermöglichen oder womöglich eine Verwechslung mit einer anderen Handgeste auftreten kann.
Aus diesem Grund wurde ebenfalls die Handgeste *Faust* hinzugenommen. Vergleicht man die beiden Handgesten in Abbildung 18 unter b) und c) erkennt man, dass diese auf den ersten Blick zwar sehr unterschiedlich zu sein scheinen, aber dennoch einige nicht unwesentliche Gemeinsamkeiten aufweisen. Da das Matching Verfahren nicht explizit nach der Form einer Hand sucht, sondern den gesamten Graphen mit der gegebenen Punktwolke vergleicht, könnte es zu einer Verwechslung von *Greifen* und *Faust* kommen. Betrachtet man die Abbildung der Faust unter c) erkennt man, dass, zusätzlich zur Hand, ein Teil des Unterarms zur Erstellung des Graphen hinzugezogen wurde. Da eine Faust keine große Fläche hat, wurde die Aufnahme explizit so gewählt, um den Unterschied zur Handgeste *Daumen*, aber auch zu *Greifen*, zu erhöhen. Nachteilig kann sich das jedoch auswirken, wenn zum Beispiel beim Versuch einen Daumen oder eine greifende Hand zu erkennen ein Großteil des Arms aufgenommen wird. Die Ähnlichkeit zur Faust könnte somit aus Sicht des Matching Algorithmus größer ausfallen als beabsichtigt. Die Kombination der drei in Abbildung 18 dargestellten Handgesten stellt somit eine gute Wahl dar, um die Robustheit des Algorithmus zu prüfen.

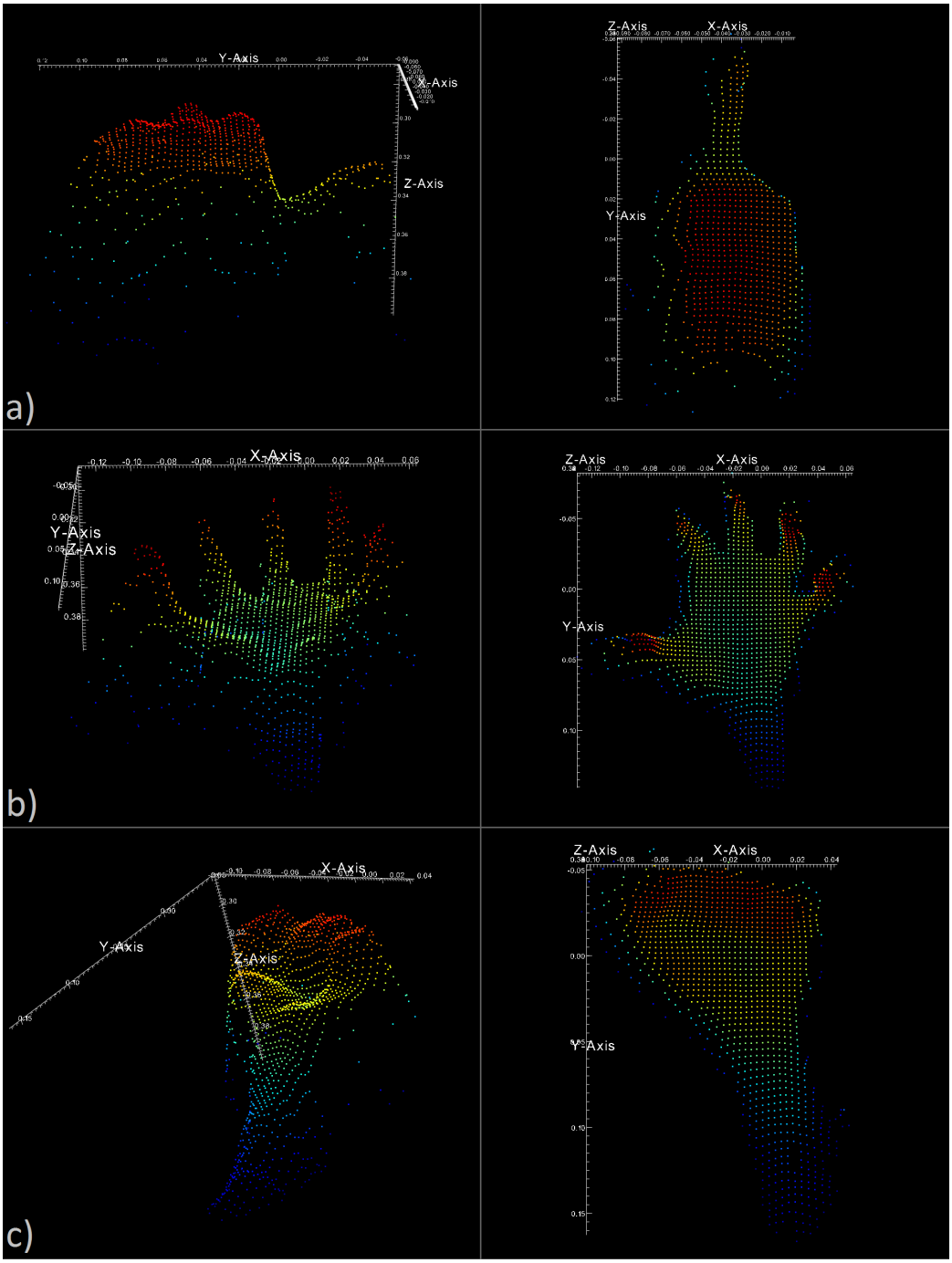

Abbildung 18: Übersicht über die Handgesten *Daumen* (a), *Greifen* (b) und *Faust* (c). Links ist jeweils eine schräge Seitenansicht zu sehen und rechts eine Draufsicht.

Mit Blick auf die bereits vorgestellten erscheint die Geste *OK* mitunter exzentrisch. Der in Abbildung 19 unter a) dargestellte Graph bildet eine Handgeste nach, bei der Daumen und Zeigefinger zu einem Kreis geschlossen sind, während die restlichen Finger leicht gekrümmt nach oben zeigen. Aufgenommen ist sie aus einer seitlichen Perspektive entlang der unteren Handkante. Diese Geste hat in vielen Ländern der Welt durchaus unterschiedliche Interpretationen, die mitunter positiv, als auch negativ ausfallen können. Aus diesem Grund sei an dieser Stelle darauf hingewiesen, dass ausschließlich auf die hierzulande gängige Bedeutung *Okay* Bezug genommen wird.
Gewählt wurde diese Handgeste, um das Verhalten bei großen Graphen und Punktwolken testen zu können. Erkennbar ist, dass ein großer Teil des Unterarms in den Graphen eingeflossen ist. Zusätzlich ist auch Interessant, wie gut die Transformation des Graphen diesen über eine gegebene Punktwolke legt, sodass das, durch den Zeigefinger und den Daumen entstandene Loch überlappt. Die Größe des Graphen kann sich hinsichtlich der Laufzeit aber auch Nachteilig auf die Erkennung auswirken.
Die letzte, im Rahmen der Testläufe, verwendete Handgeste ist *Victory*. Diese bildet eine Hand ab, bei der Zeige- und Mittelfinger ausgestreckt eine V-Form ergeben. Der kleine und der Ringfinger werden vom Daumen zur Handfläche hin gedrückt (Abbildung 19 b)). Diese ähnelt im direkten Vergleich zunächst keiner der vorgestellten Handgesten, könnte durch die Skalierung aber dennoch mit *Hand Vorne*, *Hand Hinten* oder *Hand Vorne Finger Zusammen* verwechselt werden. Trotzdem sollte ihre individuelle Form ausreichen, um eine gute Erkennung zu ermöglichen.

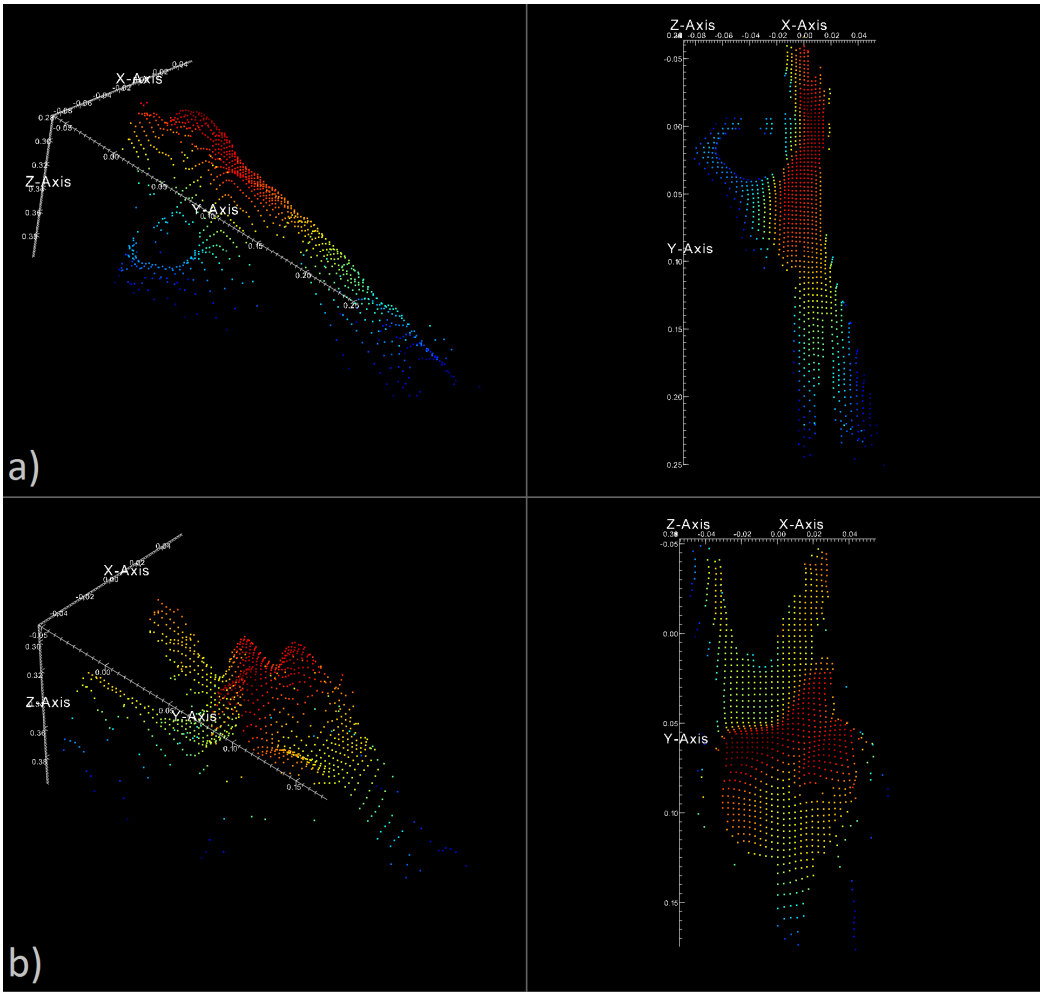

Abbildung 19: Übersicht über die Handgesten *OK* (a) und *Victory* (b).

6.4 Matchingphase

Für das Matching der Handgesten wird im Grunde der gleiche Aufbau verwendet, wie für die Aufnahme der Graphen. Die Kameraposition und die Beleuchtung bleiben unverändert. Im *Adaf*-Entwicklungsframework wird lediglich das Plugin *Graph Matching* geladen (in Abbildung 14 rot markiert), welches die beiden zuvor verwendeten Plugins *PointCloud k-Nearest Neighbor Construction* und *Graph XML Converter* ersetzt. Die restlichen zuvor verwendeten Plugins (in Abbildung 14 gelb markiert) bleiben hingegen erhalten. In *Graph Matching* lassen sich sowohl der Speicherort der zu ladenden Graphen, als auch der zu speichernden Ergebnisse festlegen. Ansonsten bedarf

das Plugin keiner weiteren Konfiguration.
Auch der Ablauf der Handgestenerkennung läuft fast genauso ab, wie die Erzeugung der Graphen. Wurde das Programm gestartet, muss nur noch die entsprechende Handgeste vor die Kamera gehalten werden. Die Punktwolke des Graphen der erkannten Handgeste wird, zusammen mit der Punktwolke der aktuell aufgenommenen Handgeste, in *Adaf* dargestellt. Um sie leichter unterscheiden zu können, sind diese farblich voneinander getrennt und sollten im Idealfall möglichst überlappen. Im *Log View* des Frameworks wird zur Laufzeit zusätzlich eine Meldung ausgegeben, die angibt welche Geste erkannt wurde und wie hoch der durchschnittliche Fehler ist. Während das Programm aktiv ist, werden die aufgenommenen Handgesten fortlaufend mit den gespeicherten Graphen verglichen und das Ergebnis ausgegeben.

6.5 Testreihen

Um die Qualität des vorgestellten Matching Verfahrens quantitativ beschreiben zu können, wurden verschiedene Testreihen durchgeführt, die einen direkten Vergleich der einzelnen Handgesten und Einstellungen ermöglichen sollen. Wie zuvor in Abschnitt 6.2 gezeigt, lässt sich beim Generieren der Graphen nicht nur die Größe der Nachbarschaften, sondern auch der Schwellwert für die Distanz zwischen den Nachbarpunkten festlegen. Auf Grundlage dessen wurden die Testreihen mit unterschiedlichen Kombinationen von Nachbarschaften und Schwellwerten durchgeführt und deren Ergebnisse tabellarisch festgehalten.
Vorangegangene Versuche zeigten, dass sich vor allem Nachbarschaften mit einer Größe von zwei, vier und acht Nachbarpunkten für die Testreihen eignen. Das Ausschlaggebende war an der Stelle nicht nur die Beschaffenheit der Graphen, sondern auch die Laufzeit während ihrer Erstellung und vor allem während der Erkennungsphase. Nachbarschaften mit mehr als acht Nachbarn führten in der Regel zu einem enormen Geschwindigkeitsverlust, der sich bei einer großen Menge an Graphen stark bemerkbar machte. Wohingegen eine Nachbarschaftsgröße von acht Punkten zwar bereits auch zu einer leichten Verminderung der Geschwindigkeit führte, während der vorangegangenen Versuche aber bereits vielversprechende Resultate lieferte und somit in die Testreihen übernommen wurde.
Für die Größe des Distanzschwellwertes (*Distance Threshold*) ergaben sich zwei Werte, die zusammen mit den verschiedenen Nachbarschaftsgrößen zu insgesamt sechs unterschiedlichen Kombinationen führten. Einerseits wurde der Wert 0.005 gewählt, was im Schnitt zu recht gleichmäßigen und lückenlosen Graphen führte, und andererseits auch der Wert 0.002, was folglicherweise zu geringeren Graphendichten und somit zu einer geringeren Kanten- und

Punktmenge für jeden Graphen führte. Die beiden Werte sollten Aufschluss darüber geben, inwiefern die Dichte der verwendeten Graphen das Ergebnis beeinflussen kann und ob es sich positiv oder negativ auf die Erkennung auswirkt.
Anzumerken sei an dieser Stelle, dass keine Einheit für den Distanzschwellwert oder die durchschnittlichen Distanzen benannt werden kann, da das *Adaf*-Framework an keiner Stelle eine Aussage dazu trifft. Das Koordinatensystem in Abbildung 20 veranschaulicht diese Problematik. Die Distanzen innerhalb der Graphen oder der Punktwolken berechnen sich vielmehr aus den Koordinatendaten der betrachteten Punkte, die sich nach diesem einheitenlosen Koordinatensystem richten.

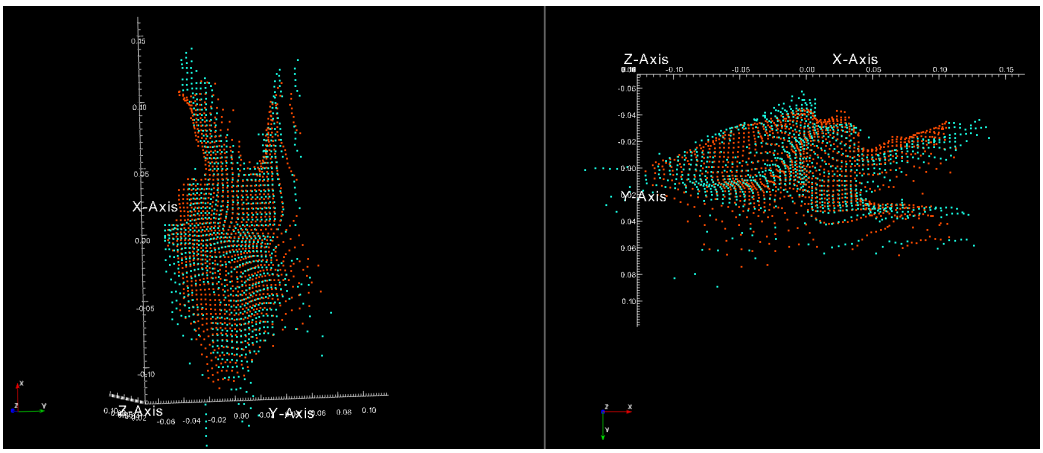

Abbildung 20: Beispiel eines Matchings: Erkannt wurde die Handgeste *Victory* (blaue Punktwolke), welche auch tatsächlich aufgenommen wurde (orangefarbene Punktwolke).

Tabelle 1 veranschaulicht die Unterschiede zwischen den Graphen für die beiden Distanzschwellwerte anhand der Handgesten *Daumen*, *Hand Vorne* und *OK* mit einer Nachbarschaftsgröße von vier. Erkennbar ist, dass die Graphen, die mit einem Distanzschwellwert von 0.002 aufgenommen wurden, eine geringere Zahl an Punkten und Kanten aufweisen, als jene, die mit dem Wert 0.005 erstellt wurden. Folglich sollte die Verwendung eines geringeren Distanzschwellwertes auch zu einer höheren Geschwindigkeit während der Erkennung führen, da pro Graph eine geringere Anzahl an Punkten verarbeitet werden muss.
Tabelle 2 gibt Aufschluss über die durchschnittlichen Laufzeiten für die einzelnen Handgesten in Kombination mit den beiden genannten Distanzschwellwerten, die sich für das verwendete Testsystem ergeben. Gemessen wird

Handgeste	Distanz	Anzahl Punkte	Anzahl Kanten
Daumen	0.002	544	856
Daumen	0.005	795	1204
Hand Vorne	0.002	1273	1924
Hand Vorne	0.005	1751	2731
OK	0.002	1004	1586
OK	0.005	1317	2047

Tabelle 1: Vergleich der Graphendichten für die beiden Distanzschwellwerte 0.002 und 0.005 bei einer Nachbarschaft von 4 am Beispiel der Handgesten *Daumen*, *Hand Vorne*, und *OK*.

anhand der *Data Package Frequency*, die das *Adaf*-Entwicklungsframework während der Handgestenerkennung bereitstellt. Dieser Wert gibt an, wie viele Durchläufe pro Sekunde stattfinden. Je höher der Wert ausfällt, desto schneller läuft dementsprechend auch die Erkennung.
Gut sichtbar ist, dass die Durchlaufzeiten stark von der jeweils zu erkennenden Handgeste abhängen. Genauer gesagt von der daraus resultierenden Größe der Punktwolke. Während die Durchlaufzeiten für die Geste *Daumen* bei einem Distanzschwellwert von 0.002 mit durchschnittlich $9 - 10$ Durchläufen pro Sekunde noch relativ hoch ausfallen, ergeben sich zum Beispiel für *OK* nur noch Werte von $5.5 - 6$. Im Vergleich dazu fallen die Durchlaufzeiten für Graphen, die mit einem Distanzschwellwert von 0.005 erstellt wurden noch weiter ab. Hier liegt die Anzahl an Durchläufen pro Sekunde für die Handgeste *Daumen* nur noch bei $7.5 - 8$ und für die Handgeste *OK* sogar nur noch bei $3.5 - 4$. Im Durchschnitt ergeben sich Laufzeiten von $6.9 - 7.4$ Durchläufen pro Sekunde für einen Distanzschwellwert von 0.002, während diese für den Wert 0.005 auf nur noch $5.1 - 6.2$ abfallen. Die Annahme, dass geringere Graphendichten zu einer höheren Durchlaufrate führen, bestätigt sich somit also.

Die Wahl eines geringeren Distanzschwellwertes liegt aber nicht nur in einer bessere Laufzeit begründet. Ein weiterer entscheidender Faktor, dem im Rahmen der Testläufe auf den Grund gegangen wird, ist die Frage ob es sich auch positiv auf die Erkennung der Handgesten auswirkt. Die im nächsten Abschnitt folgende Auswertung der Ergebnisse soll Aufschluss hierüber geben.

	Distanzschwellwert	
Handgeste (4nnk)	*0.002*	*0.005*
Daumen	$9 - 10$	$7.5 - 8$
Faust	$6.5 - 7$	$4.5 - 5$
Greifen	$8.5 - 9$	$8 - 8.5$
Hand Hinten	$6 - 6.5$	$4 - 4.5$
Hand Vorne	$6 - 6.5$	$3.5 - 4.5$
Finger Zus.	$6 - 6.5$	$4 - 4.5$
OK	$5.5 - 6$	$3.5 - 4$
Victory	$7.5 - 8$	$5.5 - 6$
$\varnothing$	$6.9 - 7.4$	$5.1 - 6.2$

Tabelle 2: Durchschnittliche Durchlaufzeiten während der Handgestenerkennung für die Distanzschwellwerte 0.002 und 0.005 mit Nachbarschaften von 4nnk (pro Handgeste und im Gesamtdurchschnitt). Die Werte geben die Anzahl der Durchläufe pro Sekunde für den Matching Algorithmus innerhalb des *Adaf*-Frameworks an. Die Durchschnittswerte sind auf eine Nachkommastelle gerundet.

6.5.1 Auswertung - Allgemein

Die Ergebnisse der Testreihen lassen sich nur mit Mühe anhand von Zahlen darstellen. Aufgrund der Arbeitsweise des Erkennungsverfahrens ist es zwar möglich den durchschnittlichen Fehler bei der Erkennung darzustellen, dieser sagt in einigen Fällen aber nur bedingt etwas über den tatsächlichen Matching-Erfolg oder -Misserfolg zur Laufzeit des Programms aus. Um diesen Umstand erklären zu können, muss an dieser Stelle aber zunächst einmal auf die Repräsentation der Testdaten eingegangen werden.
Diese sind in den Tabellen 3 bis 13 abgebildet, wobei der Aufbau sämtlicher Tabellen der Lesbarkeit halber konsistent gehalten wurde. Jede Tabelle enthält die Datensätze aller Testläufe für eine einzelne Handgeste und einen der beiden Distanzschwellwerte.
Zu Vergleichszwecken sei hier auf Tabelle 4 verwiesen, die die Daten der Handgeste *Faust* mit Distanzschwellwert 0.002 enthält. Betrachtet man die Ergebnisse der einzelnen Durchläufe stellt man fest, dass sich die Werte mitunter stark ähneln. Tatsächlich unterscheiden sich diese in den meisten Fällen erst ab der vierten Nachkommastelle. Das liegt daran, dass für die Messung der Ähnlichkeit die durchschnittliche Distanz zwischen der betrachteten Punktwolke und dem jeweiligen Graphen zum Einsatz kommt. Diese Werte fallen aufgrund der bereits vorgestellten Transformationen (Abschnitt 4.1)

extrem ähnlich aus. Genauer gesagt ist die Ursache hierfür die vorgenommene Skalierung, die die Graphen an die Größe der Punktwolke anpasst. Dabei werden die Proportionen des Graphen außer Acht gelassen. Zusätzlich bewirkt die Rotation und Translation auf Grundlage der *PCA*, dass die Graphen so gut es geht an die Punktwolke angepasst werden. Graphen, die im Grunde kaum eine Ähnlichkeit mit der betrachteten Punktwolke haben, können so unter Umständen sehr nah an die Proportionen ebendieser heranreichen.
Das hat allerdings nicht zur Folge, dass das Erkennungsverfahren schlechte Ergebnisse liefert, da die Unterschiede in den Werten nach der vierten Nachkommastelle bereits die entscheidenden Differenzierungen liefern. Beim Vergleich der Werte muss der Schwerpunkt dementsprechend auf die hinteren Ziffern gelegt werden, um sich eine Vorstellung von der Qualität des Ergebnisses machen zu können.
Betrachtet man weiterhin die Tabelle sieht man, dass die Spalte der aktuell zu analysierenden Handgeste grün hervorgehoben ist, um sie deutlich von den restlichen Spalten abzuheben. Bei Durchläufen, die pro Handgeste mehrere Graphen haben, wie zum Beispiel *2nnk - Doppelt*, ist der jeweils kleinste Wert zusätzlich fett hervorgehoben. Werte anderer Handgesten, die nahe an den kleinsten Wert der betrachteten Handgeste kommen sind gelb markiert, wobei der kleinste Wert bei Durchläufen mit mehreren Graphen ebenfalls fett hervorgehoben ist. Diese farbliche Markierung muss nicht direkt bedeuten, dass die Erkennung an dieser Stelle falsch verlief, sondern soll nur auf die Ähnlichkeit der Werte hindeuten. Eine Steigerung hierzu ist ebenfalls in Tabelle 4 zu sehen. Sollte es während der Versuche zu vermehrten Falscherkennungen gekommen sein - was meistens auch mit extrem ähnlichen Werten einhergeht - so ist das jeweilige Feld in der Tabelle rot markiert. Auf die Bedeutung der jeweiligen Stellen wird in der Auswertung der einzelnen Tabellen genauer eingegangen.
Da im Zuge der Handgestenerkennung nicht mit starren Objekten wie zum Beispiel Tassen, Bällen oder ähnlichem gearbeitet wird, sondern mit beweglichen Händen, die nur schwer in fest definierten Positionen fixiert werden können, lassen sich die Testläufe (wie zum Beispiel *2nnk* und *4nnk*) nicht direkt anhand ihrer Werte miteinander vergleichen. Worauf es vielmehr ankommt sind die relativen Abstände der Werte innerhalb einzelner Testläufe. Je geringer die Differenz des Gewinnergraphen zu einem Graphen einer anderen Handgeste, die nicht korrekt ist, umso wahrscheinlicher ist es auch, dass zur Laufzeit des Programms falsche Handgesten erkannt wurden.
Für die Aufnahme der Testdaten wurden absichtlich nur die Werte von Durchläufen mit korrekter Erkennung übernommen. Anhand dieser Daten wird dementsprechend nicht verglichen, ob die Erkennung selbst zu die-

sem Zeitpunkt korrekt verlief, sondern wie hoch die Wahrscheinlichkeit einer Falscherkennung für die betrachtete Handgeste in der jeweiligen Testreihe ist. Diese Methode erlaubt es alle Durchläufe in einen direkten Vergleich miteinander zu stellen und deren mögliche Erfolgsrate zu quantifizieren.
Es wurden insgesamt elf Tabellen zu den acht verschiedenen Handgesten erstellt - acht mit einem Distanzschwellwert von 0.002 und drei mit einem Distanzschwellwert von 0.005. Letztere sollen als Vergleich dienen, um die Unterschiede zwischen den beiden Distanzschwellwerten zu untersuchen. Deswegen werden für diesen Wert auch nur drei repräsentative Handgesten ausgewertet. Für den Wert 0.002 werden hingegen alle acht in 6.3 vorgestellten Handgesten getestet. Im Folgenden wird nun einzeln auf jede Handgeste eingegangen und deren Datensätze analysiert. Zusätzlich werden auch die aufgestellten Annahmen über die einzelnen Handgesten geprüft und es wird geschaut inwiefern sich diese bestätigen oder nicht bestätigen.

6.5.2 Auswertung - Daumen

Für die Handgeste *Daumen* wurden sowohl Durchläufe mit einem Distanzschwellwert von 0.002 als auch von 0.005 durchgeführt. Diese sind in den Tabellen 3 und 11 aufgeführt. Betrachtet man die erste Tabelle, zeichnet sich ein recht konstantes Bild für die verschiedenen Durchläufe ab. Die in den drei obersten Zeilen dargestellten Durchläufe mit Nachbarschaften von zwei, vier und acht Nachbarpunkten und jeweils einem Graphen pro Handgeste (*2nnk - Einzel*, *4nnk - Einzel* und *8nnk - Einzel*) weisen für alle verglichenen Handgesten eine, relativ betrachtet, hohe Differenz zur Gewinnerhandgeste auf. Zum Beispiel weist die Handgeste *Faust* für den Durchlauf *2nnk - Einzel* eine Differenz von 0.000290 auf. Dies trifft auch für die restlichen Werte der drei Durchläufe zu.
An dieser Stelle sei vermerkt, dass der Lesbarkeit halber davon ausgegangen werden kann, dass Differenzen über einem Wert von ungefähr 0.000100 auf eine geringe Falscherkennung während der Laufzeit des Programms hinweisen. Werte, die darunter liegen werden dementsprechend gelb in der Tabelle markiert, falls zur Laufzeit Falscherkennungen auftraten und Werte unterhalb einer Differenz von ungefähr 0.000500 werden sinngemäß rot markiert, falls diese Handgeste zur Laufzeit zu sehr häufigen Falscherkennungen geführt hat.
Die Erkennung der Handgeste *Daumen* verläuft bei allen Durchläufen mit nur einem Graphen pro Geste sehr stabil. Es kommt nur in Ausnahmefällen zu einer falschen Erkennung, die in diesen Fällen jedoch auf einen falschen Winkel der Hand zur Kamera zurückzuführen sind. Selbst über unterschiedliche Distanzen hinweg klappt die Erkennung problemlos. Aufgrund der geringen

Menge an Punkten in der aufgenommen Punktwolke kann es allerdings dazu kommen, dass der durchschnittliche Fehler zu groß wird und unter den vom Plugin vorgegebenen Schwellwert für die Erkennung fällt. Dies geschieht jedoch nur, wenn die Distanz zur Kamera zu groß wird. Die in 6.3 gestellte Annahme, dass die Lage des Daumens Einfluss auf die *PCA* und somit die Transformation haben könnte, bestätigte sich während der Testreihen aber weitestgehend nicht.
Die vierte Spalte von Tabelle 3 zeigt die Werte für den Durchlauf mit allen drei Nachbarschaften zusammen. Bei dieser Testreihe gibt es pro Handgeste insgesamt drei Graphen. Damit soll überprüft werden, ob es eine Verbesserung in der Erkennung gibt, wenn die unterschiedlichen Nachbarschaften gleichzeitig zum Einsatz kommen und ob diese den damit verbundenen Leistungsverlust rechtfertigt. Auf den ersten Blick fällt auf, dass die Werte für die Handgeste *Faust* und *Victory* zum Teil sehr nah an die Werte von *Daumen* heranreichen. Zur Laufzeit des Programms kommt es für diese Testreihe tatsächlich etwas öfter zu einer Falscherkennung, als bei den anderen drei Testreihen. Diese treten allerdings nur dann auf, wenn sich die Hand während der Aufnahme in einer schrägen Position zur Kamera befindet. Anders als angenommen stellt sich diese Testreihe im Vergleich zu den anderen dreien damit als geringfügig weniger robust dar.
Vergleicht man diese vier Testreihen für den Distanzschwellwert 0.002 nun mit den Ergebnissen der gleichen Testreihen für den Schwellwert 0.005 in Tabelle 11 stellt man fest, dass die Werte für die Einzelläufe gleichermaßen konstant ausfallen. Einzig die Testreihe, welche die drei verschiedenen Nachbarschaften vereint, weist in diesem Fall ein minimal besseres Ergebnis auf. Praktisch lassen sich zur Laufzeit allerdings kaum Unterschiede bei der Erkennung feststellen. In beiden Fällen liefern die Testreihen vergleichbare Ergebnisse. Lediglich die Durchlaufzeiten fallen, wie bereits in Tabelle 2 gezeigt wurde, geringfügig ab.
Der zweite Teil der Tabelle 3 befasst sich mit einer ähnlichen Aufstellung von Testreihen, die aber mit der doppelten Menge an Graphen pro Handgeste durchgeführt wurden. Die ersten drei Testreihen fanden mit jeweils zwei Graphen pro Handgeste statt und die letzte Testreihe vereint diese wieder zu einer großen Testreihe. Auch hier soll geprüft werden, ob die Vergrößerung der Graphenmenge einen positiven Einfluss auf die Handgestenerkennung hat - in diesem Fall jedoch mit jeweils gleichen Nachbarschaften.
Betrachtet man die Werte der ersten drei Testreihen (Spalten 5 - 7) zeichnet sich ein ähnliches Bild ab, wie für die ersten drei Testreihen zuvor. Die Differenz von *Faust* zu *Daumen* liegt im Vergleich zum gewinnenden Graphen bei $0.000240 - 0.000306$ und somit in einem vergleichbaren Intervall wie zuvor. Auch die letzte Testreihe (*Alle - Doppelt*) zeigt sehr vergleichbare Resultate.

Im Gegensatz zur Testreihe *Alle - Einzel* kommt es hier allerdings zu keinen Annäherungen der Werte an den des Gewinnergraphen. Zieht man jedoch in Betracht, dass im Rahmen der letzten Testreihe insgesamt sechs Graphen pro Handgeste zum Einsatz kommen, wird klar, dass der Leistungsverlust bei einer solch hohen Menge an Graphen enorm sein muss. Das bestätigt sich zur Laufzeit des Programms auch. Da die Anzahl an Durchläufen pro Sekunde bei der letzten Testreihe bereits bei der Handgeste *Daumen* unter 1 fällt, stellt sich diese Testreihe als reiner Sonderfall dar, der nur zu Verlgeichszwecken und der Vollständigkeit halber in die Testreihen aufgenommen wird.
Abbildung 21 veranschaulicht das Matching der Handgeste *Daumen* zur Laufzeit des Programms. Proportionen und Größe stimmen weitestgehend überein, auch wenn sie nicht perfekt sind.

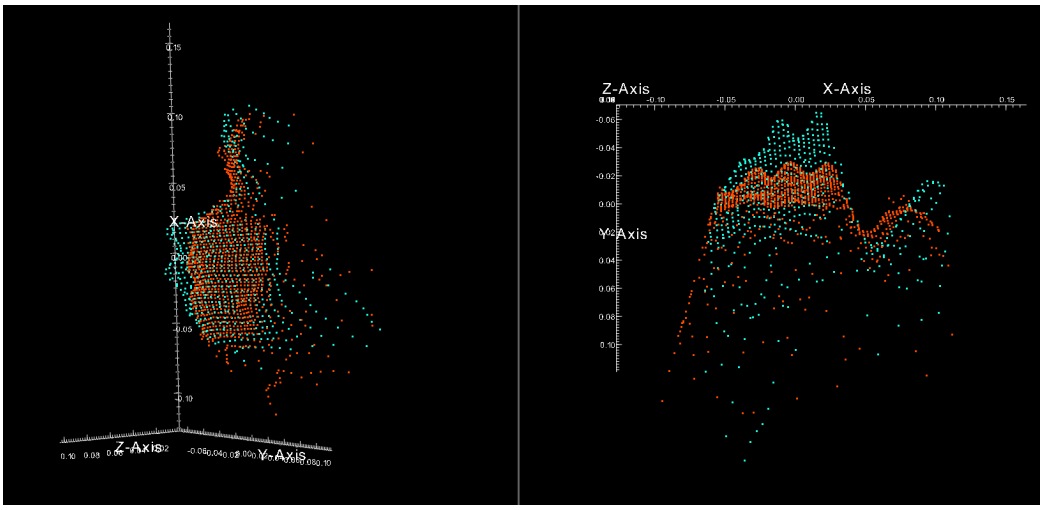

Abbildung 21: Trotz leichter Abweichungen bei der Skalierung wird die Handgeste *Daumen* sehr gut erkannt.

6.5.3 Auswertung - Faust

Die Ergebnisse der Handgeste *Faust* liegen nur für den Distanzschwellwert 0.002 vor, da die vergleichbaren Ergebnisse für 0.005 ein ähnliches Bild liefern. Zu finden sind diese in Tabelle 4.
Vergleicht man die Werte der Testreihe *2nnk - Einzel* stellt man fest, dass diese sehr eng beieinander liegen und zudem sehr nah an den Wert der erkannten Handgeste reichen. Zum Teil betragen die Differenzen weniger als 0.000050. Dennoch wurde an dieser Stelle davon abgesehen die entsprechenden Felder rot zu markieren. Zur Laufzeit des Programms zeigt sich, dass

die Erkennung weitestgehend stabil verläuft, sofern die Aufnahme im richtigen Winkel stattfindet und die Distanz zur Kamera nicht zu sehr schwankt. Aufgrund der Größe der Punktwolke und des Graphen kommt es leicht zu Verzerrungen bei der Transformation sobald die Hand zu nah an die Kamera gehalten wird. Zu Problemen kommt es auch, wenn die Hand - oder in diesem Fall der Arm - rotiert wird. Abbildung 22 veranschaulicht die genannten Situationen. Links ist zu erkennen, dass zwar die richtige Handgeste erkannt wurde, der entsprechende Graph jedoch gespiegelt ist. Zu diesem Verhalten kommt es für *2nnk - Einzel* in vermehrten Fällen. Auch wenn die richtige Distanz und der richtige Winkel eingehalten werden, scheint es zu Schwankungen zu kommen, die die *PCA* und somit die Transformation beeinflussen können. Die Regel ist dieses Verhalten aber nicht. Der rechte Teil der Abbildung verdeutlicht das Verhalten im Falle eines falsch ausgerichteten Arms. Die *PCA* und die zusätzliche Berechnung der Vorzeichen für jede Dimension (Abschnitt 4.1.4) soll zwar dafür sorgen, dass ebendies verhindert wird, scheint in manchen Fällen aber zu versagen. Man kann gut erkennen, dass statt der *Faust* die Handgeste *Hand Vorne* als möglicher Gewinner der Erkennung ermittelt wurde. Zudem ist der Graph auch horizontal gespiegelt. Die einzelnen Finger der erkannten Handgeste (blaue Punktwolke) lassen sich am unteren Ende der aufgenommenen Punktwolke (orange) ausmachen. Darüber hinaus ist der Graph sehr stark verzerrt.
Im Vergleich dazu zeigt Abbildung 23 wie die Erkennung der richtigen Handgeste *Faust* aussehen muss. Die Proportionen und die Lage des Graphen (blau) stimmen mit denen, der Punktwolke (orange) überein.

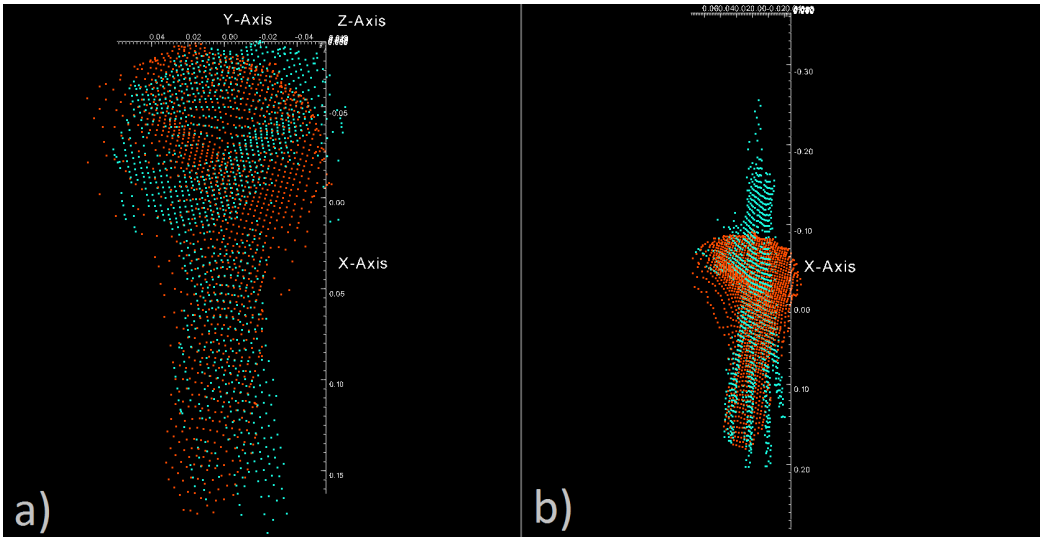

Abbildung 22: Falscherkennung der Handgeste *Faust*.

Analysiert man die Werte der Testreihen *4nnk - Einzel* und *8nnk - Einzel* scheint es so, als seien diese auf den ersten Blick besser als bei *2nnk - Einzel*. Die Differenzen der Werte liegen zum Beispiel im Vergleich zum Gewinnergraphen für *4nnk - Einzel* bei durchschnittlich 0.000100, bei *8nnk - Einzel* sogar bei 0.000150.
In der Tat bietet sich zur Laufzeit ein den Werten entsprechendes Bild. Im Schnitt ist die Erkennung etwas robuster bei *4nnk - Einzel*. Es treten zwar auch die gleichen Falscherkennungen auf, wie zuvor bei *2nnk - Einzel*, diese sind aber weitaus seltener. Hervorstechend ist, dass die Robustheit gegenüber Rotationen zugenommen hat. Auch bei falschen Aufnahmewinkeln zeigt sich die Erkennung weitaus weniger anfällig.
Eine Steigerung hierzu findet sich in den Testreihen *8nnk - Einzel*. Hier verstärkt sich der genannte Effekt noch einmal. Hinzu kommt, dass der Leistungsunterschied zwischen *2nnk - Einzel* und *4nnk - Einzel* nur geringfügig ausfällt (7.5 für *2nnk* und 7.0 für *4nnk*).
Die Testreihe *Alle - Einzel*, ist sowohl von den Ergebnissen in Tabelle 4, als auch der Erkennung zur Laufzeit, vergleichbar mit der Testreihe *4nnk - Einzel*. Die Geschwindigkeit ist aber um ein vielfaches geringer, als bei den anderen Testreihen.
Der zweite Teil der Tabelle, der sich mit den Testreihen mit doppelter Graphenmenge befasst, zeichnet ein vergleichbares Bild der Erkennung von Handgeste *Faust*. Während die Testreihe *2nnk - Doppelt* erneut einige Schwächen aufweist und die einzelnen Werte sehr nah am Wert des Gewinnergraphen

liegen, sind die Ergebnisse der Testreihen *4nnk - Doppelt*, *8nnk - Doppelt*, sowie *Alle - Doppelt* sehr stabil. Es lässt sich sogar eine deutliche Steigerung der Robustheit gegenüber der Rotation des Arms feststellen. Im direkten Vergleich ergibt sich für das Matching mit doppelter Graphenmenge für die Nachbarschaften *4nnk* und *8nnk* eine bessere Erkennung. Selbst für die Testreihe *2nnk - Doppelt* fällt die Erkennung wesentlich besser aus, als für *2nnk - Einzel*. Die Geschwindigkeit fällt allerdings auf mehr oder weniger die Hälfte der zuvor gemessenen Geschwindigkeit ab und liegt nur noch bei ungefähr 3.5 Durchläufen pro Sekunde, statt der zuvor durchschnittlich gemessenen 7 Durchläufe pro Sekunde. Die Testreihe *Alle - Doppelt* hat zwar auch solide Ergebnisse, weist aber eine so langsame Geschwindigkeit auf, dass das *Adaf*-Framework diese nicht einmal anzuzeigen vermag.
Zusammenfassend lässt sich sagen, dass sich die Handgeste *Faust* in allen Testreihen gut erkennen lässt. Während die beiden Testreihen mit Nachbarschaften von *2nnk* einige Schwächen gegenüber jeglicher Rotation aufweisen, fallen diese bei den restlichen Testreihen verhältnismäßig gering aus.

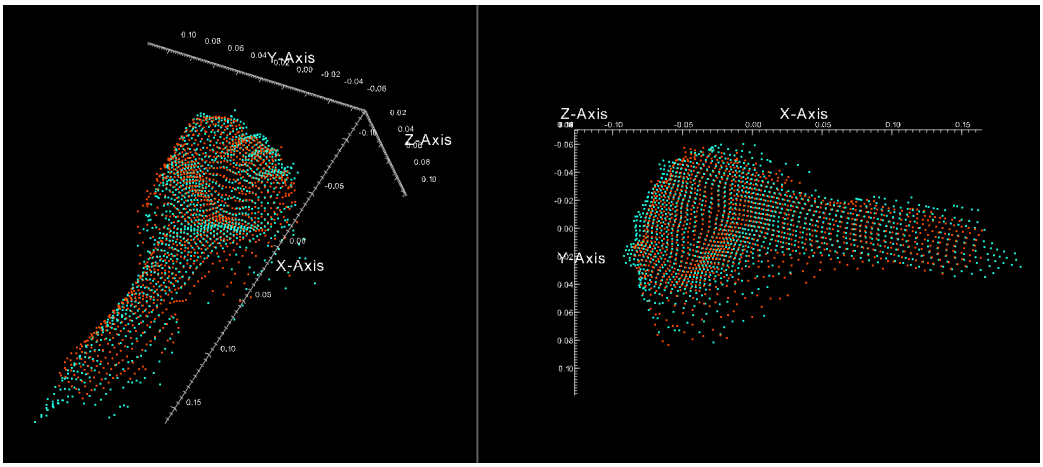

Abbildung 23: Korrekt erkannte Handgeste *Faust*.

6.5.4 Auswertung - Greifen

Die Handgeste *Greifen* führte während der Aufnahme der Testreihen zu besonderen Problemen. Auch wenn die Werte der Testreihen *2nnk - Einzel* und *4nnk - Einzel* einen anderen Schluss zulassen, stellte sich die Suche nach einer geeigneten Position der Hand für eine korrekte Erkennung als sehr schwierig heraus. Aufgrund der vorangehenden Transformation kommt es dazu, dass die Handgesten *Faust*, *HandHinten*, *HandVorne* und *Finger Zus.* mindestens

genauso oft erkannt werden, wie die zu erkennende Handgeste selbst.
Erst in der Testreihe *8nnk - Einzel* spiegelt sich dieses Verhalten auch in den Zahlen wieder. Die Differenz der Werte für die genannten Handgesten liegt im Vergleich zum Gewinnergraphen bei nur 0.000003 - 0.000055, was im Grunde zeigt, dass die Handgestenerkennung an dieser Stelle komplett indifferent ist. Um die Handgeste *Greifen* zu erkennen, ist es praktisch notwendig, die exakte Position und Ausrichtung der Hand zur Zeit der Aufnahme zu finden. Abbildung 24 verdeutlicht die problematische Lage. Unter a) ist zu erkennen, wie fälschlicherweise die Handgeste *Hand Vorne Finger Zusammen* erkannt wurde. Es fällt besonders auf, dass aufgrund der *PCA* und der damit verbundenen Skalierung eine ziemlich genaue Positionierung des Graphen durchgeführt worden ist. Sowohl die gesamte Handfläche, als auch der Daumen sind weitestgehend an der richtigen Position. Im Vergleich dazu zeigt b) wie die richtige Erkennung während der Ausführung des Programms aussieht. Die Proportionen der Hand scheinen nicht richtig zu sein, wodurch die gesamte Hand gestaucht wirkt. Es ist ebenfalls kaum erkennbar wo genau sich die Finger befinden. An dieser Stelle wird ersichtlich, warum der Algorithmus so große Probleme hat, die Handgeste *Greifen* zu erkennen.
Alle anderen Testreihen weisen, trotz verhältnismäßig guter Werte, ähnliche Schwierigkeiten in der Erkennung auf. Die Positionierung der Hand für eine korrekte Erkennung ist nahezu unmöglich. Selbst wenn die Hand regungslos vor die Kamera gehalten wird, springt die Erkennung scheinbar wahllos durch die Handgesten *Faust*, *Hand Hinten*, *Hand Vorne*, *Hand Vorne Finger Zusammen* und auch *Greifen*. Demnach bestätigen sich die Vermutungen aus 6.3 bezüglich möglicher Probleme weitestgehend und werden sogar noch übertroffen.

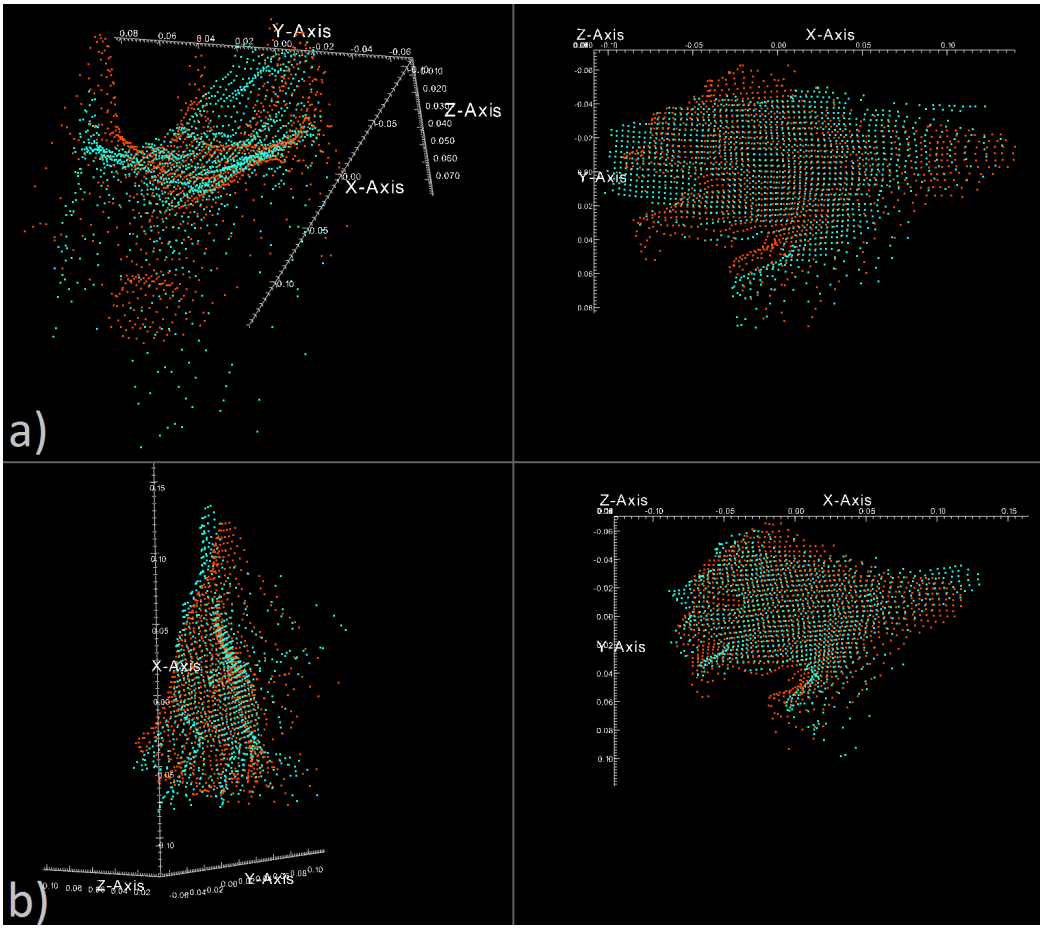

Abbildung 24: Vergleich von falsch und richtig erkannter Handgeste *Greifen*: a) zeigt die falsche Erkennung der Handgeste *Hand Vorne Finger Zusammen* und b) die korrekte Erkennung der gesuchten Handgeste *Greifen*.

Eine mögliche Lösung für dieses Problem könnte sein, eine andere Methode für die Skalierung der Graphen zu finden. Die Ausrichtung stimmt in den meisten Fällen durchaus überein, allerdings scheinen die Proportionen oft verzerrt. Ließe sich eine Skalierung finden, die eine genauere Anpassung ermöglicht, könnte dies auch zu einer besseren Erkennung der Handgeste *Greifen* führen.

6.5.5 Auswertung - Hand Hinten

Die Ergebnisse der Handgeste *Hand Hinten* liegen nur für den Distanzschwellwert 0.002 vor, obwohl sich während der Aufnahme der Testdaten zeigte, dass

es Unterschiede zu Durchläufen mit einem Distanzschwellwert von 0.005 gibt. Der Unterschied lässt sich allerdings noch besser mit der Handgeste *Hand Vorne* beschreiben, welche nachfolgend in Abschnitt 6.5.6 zu finden ist.
Betrachtet man die Zahlen in Tabelle 6 zeigt sich in einigen Testreihen, dass die Werte recht nah an die der Handgeste *Hand Vorne* reichen. In der Tat kommt es zur Laufzeit des Programms vermehrt zu Verwechslungen mit dieser Handgeste. Das ist aufgrund der recht hohen Ähnlichkeit auch nicht verwunderlich. Viel interessanter ist aber, wie oft diese Verwechslungen tatsächlich stattfinden.
Bei den Einzeldurchläufen *2nnk - Einzel*, *4nnk - Einzel* und *8nnk - Einzel* kommt es bei falscher Lage der Hand des öfteren zu einer Verwechslung mit der Handgeste *Hand Vorne*. Dies geschieht vor allem in der Testreihe mit der Nachbarschaft von *2nnk*. Nichtsdestotrotz läuft die Erkennung in den meisten Fällen richtig ab. Auch hier ist die Lage der Hand enorm wichtig. Wird zu viel vom Arm aufgenommen, beeinflusst es das Ergebnis bisweilen enorm. Außer mit der Handgeste *Hand Vorne* gibt es allerdings keine Verwechselungen mit anderen Handgesten.
In den Testreihen *4nnk - Einzel* und *8nnk - Einzel* zeigt sich während des Matchings, dass die Gefahr einer falschen Erkennung mit steigender Nachbarschaftsgröße allerdings abnimmt. Auch wenn das in Tabelle 6 für *8nnk* und die Handgeste *Hand Vorne* anders zu sein scheint.
Die Testreihe *Alle - Einzel* weist im Vergleich zu den Testreihen *4nnk - Einzel* und *8nnk - Einzel* allerdings keine Unterschiede auf und lässt sich von der Qualität der Erkennung mit diesen vergleichen.
Ganz anders sieht es bei den Testreihen mit doppelter Graphenanzahl aus. Für *2nnk - Doppelt*, *4nnk - Doppelt* und *8nnk - Doppelt* lässt sich festhalten, dass sich die Erkennung grundsätzlich verbessert. Vor allem aber wird sie weniger anfällig gegen Rotationen und wechselnde Distanzen. In einigen Fällen kommt es zwar auch hier zu wiederholten Falscherkennungen, allerdings sind diese eher die Ausnahme als die Regel.
Abbildung 25 demonstriert, warum es zu vermehrten Verwechslungen zwischen den beiden Handgesten *Hand Hinten* und *Hand Vorne* kommt. Unter a) ist die korrekte Erkennung dargestellt. Gut zu erkennen ist der runde Handrücken. Der Graph (blau) und die Punktwolke (orange) überschneiden sich an dieser Stelle ziemlich genau. Auch von oben betrachtet stimmen sowohl Proportionen, als auch Größe überein. Schaut man sich zum Vergleich an, wie die falsche Erkennung der Handgeste *Hand Vorne* aussieht, erkennt man, dass von oben betrachtet kaum ein Unterschied zu erkennen ist. Erst von der Seite zeigt sich, dass der runde Handrücken und die Kuhle der von vorne betrachteten Hand nicht übereinstimmen.
Da sowohl Rotation, als auch Skalierung hier sehr gut funktionieren, wür-

de eine andere Skalierungsmethode keine Verbesserung bringen (aber auch keine Verschlechterung). Um eine Verbesserung an dieser Stelle zu ermöglichen, müsste eine bessere Methode für den Vergleich der Graphen gefunden werden, die zum Beispiel die lokalen Strukturen des Graphen berücksichtigt.

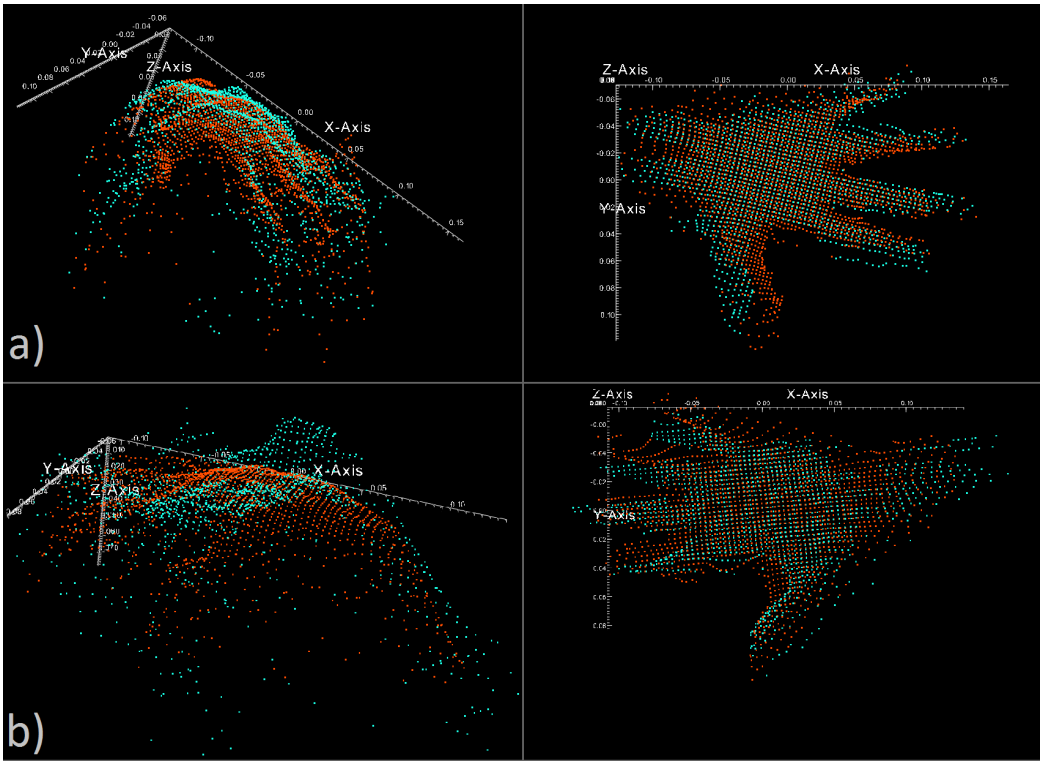

Abbildung 25: Vergleich von falsch und richtig erkannter Handgeste *Hand Hinten*: a) zeigt die richtige Erkennung und b) die falsche Erkennung der Handgeste *Hand Vorne*.

6.5.6 Auswertung - Hand Vorne

Wie bereits erwähnt wurden für die Handgeste *Hand Vorne* sowohl Testreihen für den Distanzschwellwert 0.002, als auch Testreihen für den Distanzschwellwert 0.005 durchgeführt. Die Ergebnisse sind in den Tabellen 7 und 12 festgehalten. Um einen direkten Vergleich zu erleichtern, wird zunächst nur auf die Ergebnisse der Tabelle 7 eingegangen.
In Abschnitt 6.3 wurde bereits die Vermutung aufgestellt, dass die drei Handgesten *Hand Hinten*, *Hand Vorne* und *Hand Vorne Finger Zusammmen*, aufgrund ihrer hohen Ähnlichkeiten, zu vermehrten Verwechslungen führen könnten. Das hat sich bei der Auswertung der Handgeste *Hand Hinten* auch

bereits weitestgehend zeigen können. Die Frage die sich nun stellt ist, ob diese Verwechslungen auch in der anderen Richtung auftreten. Vergleicht man in Tabelle 7 die Spalten von *Hand Hinten* und *Hand Vorne* zeigt sich interessanterweise, dass die Differenzen der beiden Handgesten durchaus groß ausfallen. Bis auf *2nnk - Einzel* scheinen die Testreihen gute Werte zu liefern. Das bestätigt sich auch zur Laufzeit des Programms. Die Handgeste *Hand Vorne* wird nur in äußerst seltenen Fällen mit der Handgeste *Hand Hinten* verwechselt.
Im Gegensatz dazu kommt es jedoch zu vermehrten Verwechslungen mit der Handgeste *Hand Vorne Finger Zusammen*. Häufig geschieht das allerdings nur, wenn die Hand rotiert oder in einem hohen Winkel zur Kamera aufgenommen wird. Dieses Verhalten spiegelt sich auch in den Ergebnissen der Testreihen wider. In sämtlichen Durchläufen liegen die Werte für die beiden Handgesten sehr nahe beieinander. Das schränkt die Erkennung allerdings kaum ein, sodass trotzdem eine gutes Matching möglich ist.
Bei den Einzeldurchläufen zeigt sich, dass vor allem die Nachbarschaften *4nnk* und *8nnk* gute Ergebnisse liefern. Gleiches zeigt sich auch für die Testreihen mit doppelter Graphenmenge, wo Nachbarschaften von *2nnk* zwar eine solidere Erkennung leisten als die Testreihen mit einzelnen Graphen, aber dennoch schlechter abschneiden, als Testreihen mit den Nachbarschaften *4nnk* und *8nnk*. Zu Verwechslungen mit anderen Handgesten kommt es für die Handgeste *Hand Vorne* hingegen so gut wie gar nicht. Die Ergebnisse in Tabelle 7 belegen das auch weitestgehend.
Die Handgeste *Hand Vorne* wurde für den Vergleich zwischen den Distanzschwellwerten 0.002 und 0.005 gewählt, weil sich anhand der erstellten Tabellen besonders gut zeigen lässt, welchen Unterschied die beiden Werte auf die Handgestenerkennung haben. Vor allem aber lassen sich die Ergebnisse ziemlich genau auf die Handgesten *Hand Hinten* und *Hand Vorne Finger Zusammen* übertragen, die ein ähnliches Erkennungsmuster aufweisen.
Tabelle 12 veranschaulicht, was zur Laufzeit der Handgestenerkennung passiert. Die Handgeste *Hand Vorne* wird sowohl mit der Handgeste *Hand Hinten*, als auch *Hand Vorne Finger Zusammen* verwechselt. Abbildung 26 visualisiert das anhand eines Beispiels. Man erkennt, dass unter b) vor allem das gebogene Handgelenk der aufgenommenen Hand (orange) zu einer großen Ähnlichkeit mit der im Graphen (blau) gespeicherten Hand führt. Die richtig erkannte Handgeste unter a) weist hingegen ein gerades Handgelenk auf. Vor allem eine Rotation der Hand während der Aufnahme führt zu dieser Ausrichtung des Handgelenks, was zu einer vermehrten Falscherkennung führen kann. Hier zeigt sich besonders, dass das Erkennungsverfahren stark von der Lage der Hand während der Aufnahme abhängig ist. Der Algorithmus kann nicht zwischen Hand und Arm unterscheiden, wodurch sich die Erkennung

verschlechtert, sobald Teile des Arms in die Aufnahme einfließen, die nicht Teil des zu matchenden Graphen sind.

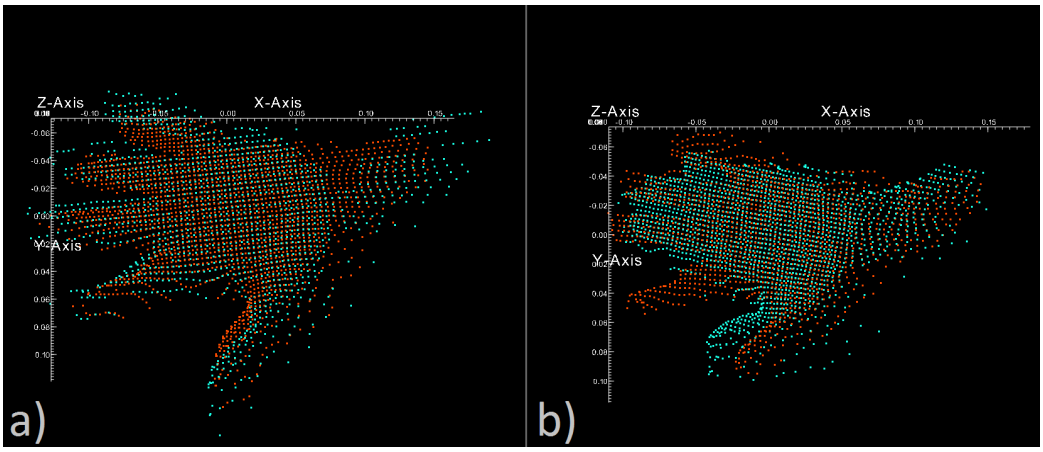

Abbildung 26: Vergleich von falsch und richtig erkannter Handgeste *Hand Vorne*: a) zeigt die richtige Erkennung und b) die falsche.

Im direkten Vergleich zu den Testreihen mit Distanzschwellwert 0.002 verschlechtert sich die Erkennung der Testreihen mit Distanzschwellwert 0.005, bleibt aber in den meisten Fällen dennoch einigermaßen robust. In einigen Fällen fällt es schwer, eine richtige Position der Hand zu finden, um ein korrektes Matching zu ermöglichen. Betrachtet man die Werte der einzelnen Durchläufe lassen sich kaum Unterschiede zwischen den einzelnen Testreihen ausmachen. Dennoch liefern die Durchläufe mit doppelter Graphenmenge erneut bessere Ergebnisse als die mit einfacher Menge. Die Durchläufe mit Nachbarschaften von *8nnk* führen, anders als zuvor, zu einer schlechteren Erkennung der richtigen Handgeste. Nachbarschaften von *2nnk* und *4nnk* sind von der Qualität der Erkennung hingegen vergleichbar.
Zieht man in Betracht, dass bei höherem Distanzschwellwert nicht nur die Qualität der Erkennung, sondern auch die Geschwindigkeit der einzelnen Durchläufe abnimmt, kommt man zum Schluss, dass ein Distanzschwellwert von 0.002 bisher besser für die Handgestenerkennung geeignet ist, als einer von 0.005.

6.5.7 Auswertung - Hand Vorne Finger Zusammen

Die Ergebnisse der Auswertung für die Handgeste *Hand Vorne* sind sehr gut auf die Auswertung der Handgeste *Hand Vorne Finger Zusammen* übertragbar. Betrachtet man die Ergebnisse in Tabelle 8 zeigt sich erneut ein ähnliches

Bild. Über alle Testreihen hinweg gibt es eine deutliche Ähnlichkeit in den Werten der beiden Handgesten. In diesem Fall sind die Differenzen sogar noch geringer als zuvor. In den Testreihen *2nnk - Einzel*, *4nnk - Einzel* und *8nnk - Einzel* liegen zudem auch die Werte für die Handgeste *Hand Hinten* sehr nah an denen von *Hand Vorne Finger Zusammen*. Zur Laufzeit zeigt sich, dass vor allem die geschlossene Handfläche während der Aufnahme dazu führt, dass die beiden Handgesten *Hand Hinten* und *Hand Vorne* fälschlicherweise erkannt werden. Aufgrund der Skalierung, welche die Graphen recht genau über die Punktwolke legt, findet der Matching Algorithmus eine große Menge an Punkten in der Punktwolke, die eine sehr geringe Distanz zu den Punkten der jeweiligen Graphen aufweisen. In 27 wird diese Problematik visualisiert. Zu erkennen ist, dass sowohl der Graph der gesuchten Handgeste gut auf die Punktwolke angepasst wird, als auch der Graph der Handgeste *Hand Vorne*. Auch wenn die Proportionen und die Lage des Graphen unter b) nicht perfekt auf die Punktwolke passt, so reicht diese Ähnlichkeit bereits aus, wenn die Handgeste nicht korrekt in die Kamera gehalten wird. Auch in diesem Fall ist die Lage des Handgelenks ausschlaggebend für die Erkennung. Vergleicht man die Punktwolke (orange) unter a) und b), kann man ein leichtes Abknicken nach rechts feststellen. Achtet man während des Matching Vorgangs auf die Lage seines Handgelenks, läuft die Erkennung der Handgeste *Hand Vorne Finger Zusammen* allerdings ohne Probleme ab.
Durchläufe mit Nachbarschaften der Größen *2nnk*, *4nnk* und *8nnk* führen, trotz der genannten Schwierigkeiten, zu einer guten Erkennung. Für Durchläufe mit gemischter Graphenmenge, wie es bei der Testreihe *Alle - Einzel* der Fall ist, kommt es etwas öfter zu Verwechslungen, als bei den anderen dreien. Die Testreihen mit doppelter Graphenmenge hingegen weisen eine Verbesserung auf, was sich in den Ergebnissen der Tabelle 8 aber nicht widerspiegelt. Der schlechte Wert für die Handgeste *Victory* in der Testreihe *2nnk - Doppelt* kann hingegen als Ausreißer betrachtet werden, da es zu keinem Zeitpunkt während der Versuche zu einer Verwechslung mit dieser Geste kam.
Zusammenfassend kann für *Hand Hinten*, *Hand Vorne* und *Hand Vorne Finger Zusammen* gesagt werden, dass es sicherlich Verbesserungsmöglichkeiten hinsichtlich des Algorithmus geben kann, die eine Verwechslung weiter minimieren könnte. Die Skalierung steigert zwar die Ähnlichkeit der Graphen und somit die Wahrscheinlichkeit einer Verwechslung, lässt sich hier aber nicht als direkte Ursache der falschen Erkennungen ausmachen. Vielmehr müsste ein Weg gefunden werden, eine Segmentierung der Hand vorzunehmen, um mögliche Störelemente, wie zum Beispiel Teile des Armes aus dem Matching Verfahren auszuschließen.

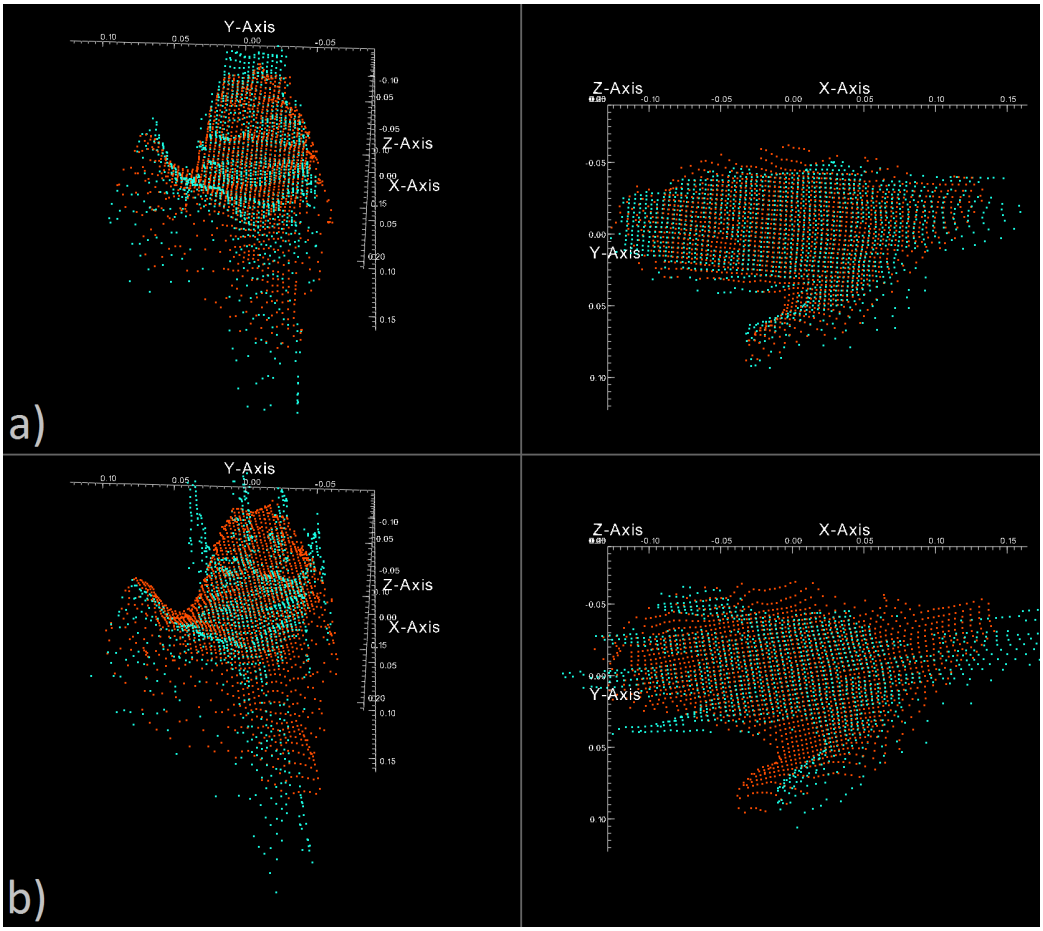

Abbildung 27: Erkennung der Handgeste *Hand Vorne Finger Zusammen*: a) zeigt die korrekte Erkennung und b) die falsche Erkennung der Handgeste *Hand Vorne*.

6.5.8 Auswertung - OK

Bei *OK* handelt es sich um die größte Handgeste von allen, die im Rahmen der Testreihen verglichen werden. Dementsprechend stellt sich die Frage, ob es zusätzlich zu den Unterschieden in den verschiedenen Testreihen, auch Unterschiede für einen variierenden Distanzschwellwert bei der Erkennung gibt. Aus diesem Grund wurde nicht nur mit dem Distanzschwellwert 0.002, sondern auch mit 0.005 getestet. Die Tabellen 9 und 13 zeigen die Ergebnisse dieser Testreihen.
Vergleicht man die Werte der beiden Tabellen für die Testreihen *2nnk - Einzel*, *4nnk - Einzel*, *8nnk - Einzel* und *Alle - Einzel* lassen sich kaum Un-

terschiede feststellen. In beiden Fällen liegen die Differenzen zwischen dem Gewinnergraphen und der jeweiligen Handgeste in einem Intervall von ungefähr $0,000150$ bis teilweise sogar $0,000400$. Während des Matchings fällt vor allem auf, dass aufgrund der Größe der Punktwolke die Geschwindigkeit deutlich im Vergleich zu den anderen Handgesten abnimmt. Tabelle 2 hat das bereits für Nachbarschaften von *4nnk* gezeigt. Während bei einem Distanzschwellwert von 0.002 durchschnittlich $5.5 - 6$ Durchläufe pro Sekunde zustande kommen, fällt die Geschwindigkeit bei einem Distanzschwellwert von 0.005 auf nur noch $3.5 - 4$ Durchläufe pro Sekunde.
Bei den Testreihen mit doppelter Graphenmenge wird der Geschwindigkeitsverlust noch deutlicher. Hier fallen die Werte nochmals auf die Hälfte ab und liegen für den Distanzschwellwert 0.002 bei ungefähr $2.5 - 3$ Durchläufen pro Sekunde und beim Distanzschwellwert von 0.005 nur noch bei $1 - 2$.
Bei der Erkennung zeigt sich die Handgeste *OK* als äußerst stabil. Aufgrund ihrer Größe und Form hebt sie sich sehr von den restlichen Handgesten ab, was sich auch während des Matchings zeigt. Zu Verwechslungen kommt es nur dann, wenn die Distanz des Arms zur Kamera während der Aufnahme zu gering ist. In diesen Fällen kommt es dann auch zu Verwechslungen mit der Handgeste *Faust*, die genauso wie *OK* einen großen Teil des Arms beinhaltet. Die erfolgreiche Erkennung ist hier vor allem der guten Transformation der Graphen geschuldet. Diese passt den Graphen der Handgeste ziemlich genau an die Punktwolke an, wie es in Abbildung 28 unter a) sichtbar ist. Trotz einer unterschiedlichen Ausrichtung der Finger bei der Punktwolke (orange), liegt der Graph (blau) an der richtigen Stelle, sodass sogar die Öffnungen von Daumen und Zeigefinger genau überlappen. Wird die Distanz zur Kamera zu gering, kann es aber passieren, dass die Skalierung den Graphen zu sehr in die Länge zieht und das zu Verwechslungen führen kann, wie unter b) zu sehen ist.

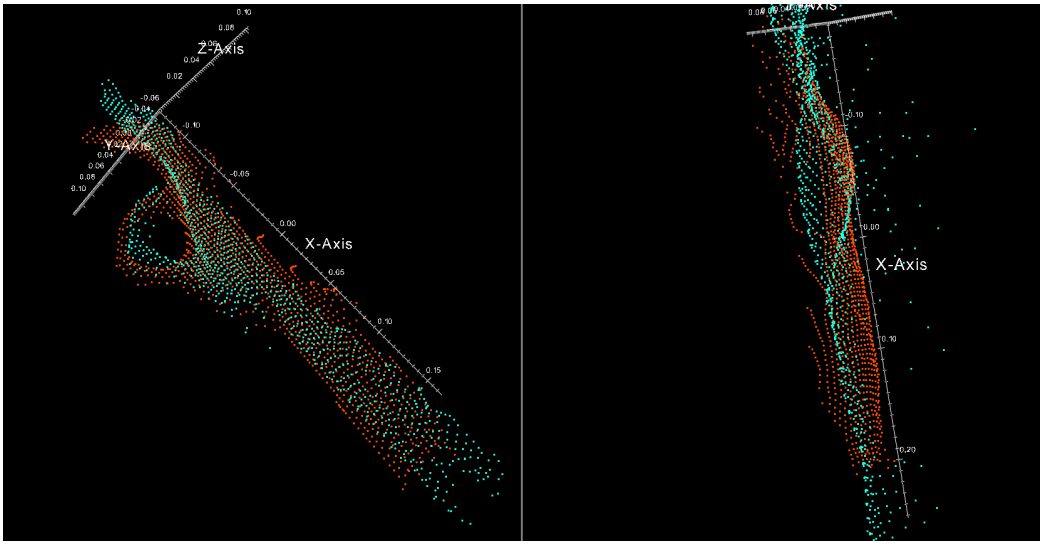

Abbildung 28: Die Handgeste *OK* während des Matchings. a) Der Graph ist sehr gut an die Punktwolke angepasst. b) Aufgrund einer zu geringen Distanz zur Kamera kommt es zu einer falschen Skalierung und somit zu Verwechslungen.

Der Vergleich der beiden Distanzschwellwerte zeigt bei der Handgeste *OK*, was bereits in Abschnitt 6.3 vermutet wurde. Aufgrund ihrer einzigartigen Form und Größe hebt sie sich deutlich von den anderen Handgesten ab. Gleichzeitig führt die Größe aber auch zu einer Reduktion der Geschwindigkeit, weswegen sich der Distanzschwellwert 0.002 als geeigneter erweist.

6.5.9 Auswertung - Victory

Victory ist die letzte Handgeste, mit der getestet wurde. Auf Testreihen mit einem Distanzschwellwert von 0.005 wurde hierbei verzichtet, da diese, bis auf eine reduzierte Geschwindigkeit während der Aufnahme, keine nennenswerten Unterschiede zum Distanzschwellwert 0.002 ergaben. Tabelle 10 fasst die Ergebnisse der Testreihen mit Distanzschwellwert 0.002 zusammen. Erkennbar ist, dass es zu leichten Annäherungen der Werte für die Handgeste *Hand Hinten* bei den Testreihen *2nnk - Einzel* und *4nnk - Einzel* kommt. Das geschieht vor allem dann, wenn ein Teil des Arms mit in die Aufnahme gerät und die Punktwolke vergrößert. Dieses Verhalten lässt sich im Grunde in allen Testreihen beobachten, was daran zu erkennen ist, dass ähnliche Annäherungen der Werte in der Testreihe *4nnk - Doppelt* zu verzeichnen sind. In den meisten Fällen wird die richtige Handgeste aber dennoch erkannt. Inter-

essanter ist jedoch, dass die transformierten Graphen in einigen Situationen horizontal gespiegelt sind und trotzdem die richtige Handgeste erkannt wird. Dieses Verhalten ist in Abbildung 29 festgehalten. Man kann unter b) erkennen, dass die aufgenommene Punktwolke (orange) auf dem Kopf steht, während der Graph (blau) richtig herum dargestellt ist. Das zeigt, dass trotz der Korrektur der Vorzeichen, wie es in Abschnitt 4.1.4 beschrieben wird, die Ausrichtungen des Graphen und der Punktwolke nicht aneinander angepasst werden konnten. Hier zeigt sich erneut, dass der in 4.2 beschriebene Algorithmus sehr empfindlich auf Veränderungen in den Details der Handgesten reagiert und diese unter Umständen falsch verarbeitet. Die transformierten Graphen der übrigen Handgesten scheinen sich hierbei aber so sehr von der Handgeste *Victory* zu unterscheiden, dass es dennoch gelingt zu einem korrekten Matching zu gelangen.

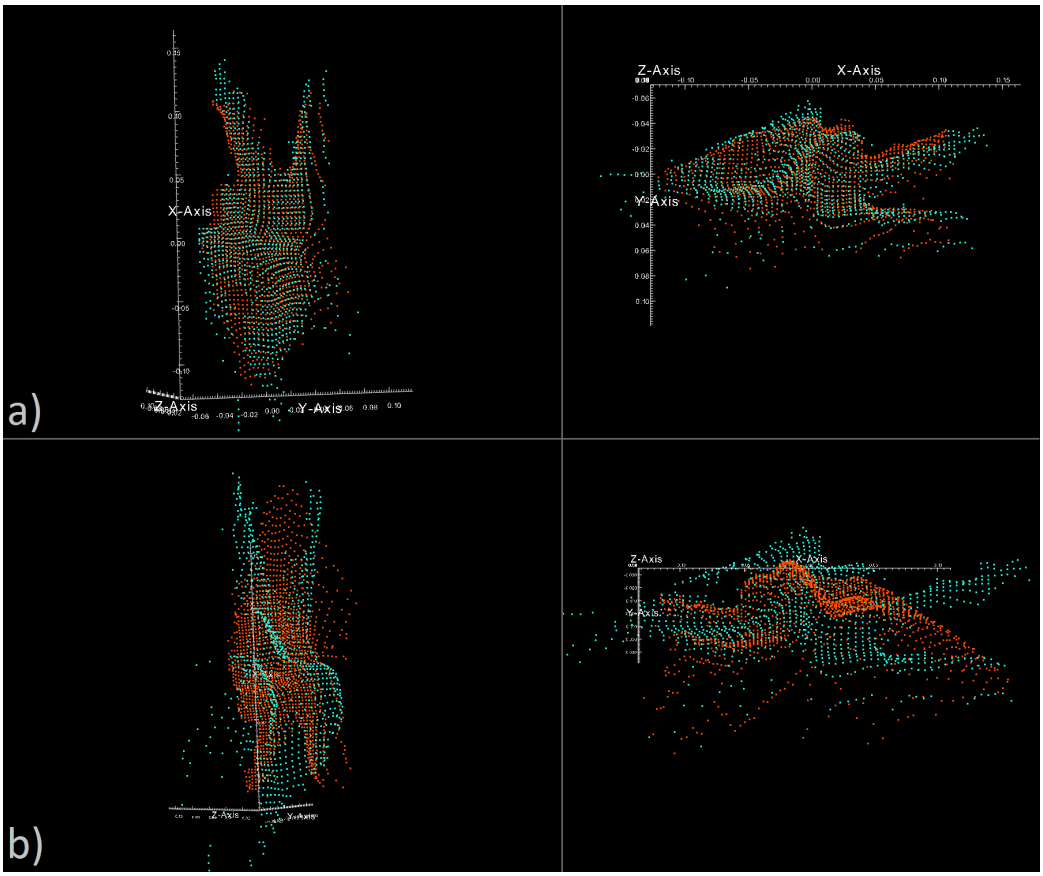

Abbildung 29: Vergleich der Matching-Ergebnisse für die Handgeste *Victory*

Während des Matching Vorgangs zeigte sich unter anderem auch, dass es erneut keine Verbesserung der Erkennung gibt, wenn - wie es in der Testreihe *Alle - Einzel* der Fall ist - alle Graphen der drei Nachbarschaftsgrößen *2nnk*, *4nnk* und *8nnk* zu einer großen Menge vereint werden.
Anders sieht es hingegen für die Testreihen mit doppelter Graphenmenge aus. Hier scheinen sich, wie zuvor auch bei vielen der anderen Handgesten, die Raten an erkannten Handgesten zu stabilisieren. Vor allem aber nimmt die Toleranz gegen Rotationen während der Aufnahme zu.

7 Fazit

Die auf 3D-Punktwolken basierende Handgestenerkennung, wie sie im Rahmen dieser Arbeit durchgeführt wurde, weist bei abschließender Betrachtung einige Vor- und Nachteile auf.
Unter anderem stellt sich die Verwendung dreidimensionaler Graphen zur Speicherung und Verarbeitung der Handgesten als äußerst effektiv heraus, da sie die großen Datenmengen der aufgenommenen Punktwolken sehr gut auf kleinere Strukturen herunterbrechen. Der hierfür verwendete *k-Nearest Neighbor* Algorithmus erweist sich dabei als sehr performante Lösung.
Dennoch lässt sich sagen, dass an dieser Stelle noch Raum für Verbesserungen besteht, da bei der Generierung der Graphen zum Beispiel nicht auf die spezifische Form der Hand geachtet wird. Würden die Graphen zum Beispiel einer abschließenden Verarbeitung unterzogen, bei der einzelne Bereiche der Hand, wie etwa die einzelnen Finger oder die Handfläche, segmentiert werden würden, könnten diese Informationen für eine tiefgreifendere Analyse während des Erkennungsverfahrens genutzt werden. Gleichzeitig könnte auf diese Weise das Problem angegangen werden, dass Graphen nach der Transformation falsch transformiert und zum Teil horizontal oder vertikal gespiegelt werden.
Trotz der genannten Schwäche erweist sich die Transformation der Graphen auf Grundlage der Hauptkomponentenanalyse allerdings als sehr robust, wenn es darum geht diese in die richtige Lage zur Punktwolke zu bringen. Wie die Auswertung der Testreihen gezeigt hat, besteht höchstens bei der verwendeten Skalierungsmethode Verbesserungsbedarf, da diese in einigen Fällen zu Verwechslungen bei der Erkennung geführt hat. Hier könnte es hilfreich sein die Proportionen nur innerhalb eines definierten Maßes verändern zu dürfen, wodurch übermäßige Streckungen oder Stauchungen der Graphen vermieden werden könnten. Das würde vor allem die Verwechslung von Handgesten mit großer Ähnlichkeit verhindern.
Der Matching Algorithmus, wie er in dieser Arbeit vorgestellt wurde, stellt eine solide Grundlage dar, um gute Resultate bei der Erkennung von Handgesten zu erhalten. Allerdings zeigen sich auch hier einige Schwächen, die vor allem auf die weitestgehend einfache Arbeitsweise zurückzuführen sind. Als nachteilig kann vor allem angesehen werden, dass der Algorithmus die Struktur der Graphen beim Vergleich mit einer aufgenommenen Punktwolke im Grunde völlig außer Acht lässt. Eine Verbesserung könnte zum Beispiel so aussehen, dass ein zu vergleichender Graph auf die Punktwolke abgebildet werden würde und die Distanzen zwischen den einzelnen Knoten des abgebildeten Graphen dann mit den Distanzen der Knoten des Ursprungsgraphen verglichen werden.
Darüber hinaus hat die Auswertung der Testreihen gezeigt, dass das momen-

tan verwendete Matching Verfahren sehr von der Lage der Hand und des Arms während der Aufnahme abhängig ist und darüber hinaus nicht unterscheiden kann, ob sich tatsächlich eine Hand vor der Kamera befindet oder nicht. Ein weiterer Schritt könnte also sein einen Algorithmus zu entwickeln, der die genannten Verbesserungen umsetzt.
Im Allgemeinen wurde aber gezeigt, dass die Verwendung von Punktwolken zur Erkennung von Handgesten durchaus zu vielversprechenden Ergebnissen führt.

A Anhang

Konfiguration	**Daumen**	Faust	Greifen	HandHinten	HandVorne	Finger Zus.	OK	Victory
2nnk - Einzel	0.346427	0.346717	0.346679	0.346639	0.346769	0.346742	0.346879	0.346528
4nnk - Einzel	0.357542	0.357713	0.357741	0.357726	0.357812	0.357811	0.357969	0.357708
8nnk - Einzel	0.32675	0.326886	0.326998	0.327018	0.32712	0.327049	0.327095	0.326825
Alle - Einzel	0.360253	0.360361	0.360344	0.360345	0.360453	0.360414	0.36056	0.360324
	0.360242	0.36028	0.360312	0.360361	0.360388	0.360415	0.360533	0.360352
	0.360241	**0.360264**	0.360317	0.360307	0.360362	0.360381	0.360508	**0.360314**
2nnk - Doppelt	**0.347327**	0.347633	0.347585	0.347556	0.347675	0.347663	0.347817	0.34748
	0.347365	0.347567	0.347559	0.347589	0.347661	0.347633	0.34781	0.347464
4nnk - Doppelt	0.356018	0.356182	0.356305	0.356269	0.356363	0.356334	0.356448	0.356153
	0.356006	0.356154	0.356305	0.356291	0.356365	0.356357	0.356429	0.356133
8nnk - Doppelt	0.384779	0.384865	0.384878	0.384888	0.384925	0.384973	0.385099	0.384896
	0.38475	0.384866	0.384893	0.384915	0.384959	0.384971	0.385123	0.384924
Alle - Doppelt	0.388174	0.388302	0.388245	0.388267	0.388357	0.388324	0.388531	0.38825
	0.38827	0.388239	0.38822	0.388279	0.388343	0.3883	0.3885	0.388222
	0.388164	0.388231	0.388214	0.388273	0.388295	0.388325	0.388478	0.388249
	0.388158	0.388213	0.388227	0.388278	0.388299	0.388321	0.38847	0.388271
	0.388155	0.388213	0.388217	0.388233	0.38827	0.3883	0.388453	0.388248
	0.388121	0.388215	0.388228	0.388258	0.388305	0.388301	0.388477	0.388258

Tabelle 3: Matching Ergebnisse für die Handgeste *Daumen* mit einem *Distance Threshold* von 0.002.

Konfiguration	Daumen	**Faust**	Greifen	HandHinten	HandVorne	Finger Zus.	OK	Victory
2nnk - Einzel	0.27714	0.277096	0.277133	0.277123	0.27714	0.277122	0.277189	0.277131
4nnk - Einzel	0.313158	0.313021	0.313149	0.313114	0.313167	0.313121	0.313281	0.313154
8nnk - Einzel	0.295806	0.295683	0.295806	0.295762	0.295798	0.295768	0.295845	0.295843
Alle - Einzel	0.256609	0.256556	0.256571	0.256539	0.256576	0.256566	0.256669	0.256592
	0.256584	**0.256464**	0.256584	0.256529	0.256594	0.256545	0.25666	0.256582
	0.256599	0.256467	0.256599	0.256545	0.256577	0.256554	0.256645	0.256673
2nnk - Doppelt	0.282533	**0.28246**	**0.282463**	**0.282417**	**0.282477**	0.282473	0.28268	**0.282492**
	0.282508	0.282365	0.28248	0.282482	0.282489	**0.28247**	**0.282633**	0.282519
4nnk - Doppelt	0.307352	**0.307253**	0.307356	0.307331	0.307388	0.307339	0.307448	0.307353
	0.307362	0.307261	0.307359	0.307332	0.307381	0.307342	0.307446	0.307339
8nnk - Doppelt	0.265253	**0.265104**	0.265235	0.265219	0.265235	0.265217	0.265341	0.265326
	0.265284	0.265108	0.265234	0.265214	0.265227	0.265209	0.265368	0.265251
Alle - Doppelt	0.330784	**0.330622**	0.3307	0.330715	0.330681	0.330707	0.330957	0.330736
	0.330788	0.330667	0.330708	0.330728	0.330692	0.330707	0.330917	0.33074
	0.330795	0.330659	0.330712	0.330711	0.330692	0.330692	0.330901	0.330724
	0.330794	0.330662	0.330709	0.330703	0.330695	0.330692	0.330895	0.330738
	0.330812	0.330657	0.330719	0.33073	0.330692	0.330699	0.330842	0.330785
	0.330831	0.330662	0.330716	0.330727	0.330687	0.330683	0.330869	0.330775

Tabelle 4: Matching Ergebnisse für die Handgeste *Faust* mit einem *Distance Threshold* von 0.002.

Konfiguration	Daumen	Faust	**Greifen**	HandHinten	HandVorne	Finger Zus.	OK	Victory
2nnk - Einzel	0.327543	0.327575	0.327383	0.327529	0.327541	0.327573	0.327925	0.327515
4nnk - Einzel	0.354825	0.354858	0.354635	0.354792	0.354786	0.354824	0.355157	0.354786
8nnk - Einzel	0.295252	0.295061	0.29501	0.295065	0.295013	0.295018	0.295352	0.295229
Alle - Einzel	0.334658	0.334701	0.3345	0.334648	0.334667	0.334693	0.335041	0.334642
	0.334651	0.334674	0.334489	0.334635	0.334637	0.33467	0.335015	0.334639
	0.334679	0.334657	**0.334485**	0.334623	0.334627	0.334657	0.334965	0.334669
2nnk - Doppelt	0.355446	0.355492	0.355325	0.355446	0.355455	0.355468	0.355877	0.355429
	0.355448	0.355516	**0.35531**	0.355412	0.355453	0.355435	0.355834	0.355427
4nnk - Doppelt	0.340428	0.340417	**0.340191**	0.340366	0.340352	0.340387	0.34073	0.340346
	0.340409	0.340383	0.340197	0.340375	0.34037	0.340386	0.340717	0.340518
8nnk - Doppelt	0.350533	0.350509	**0.350392**	0.350458	0.350445	0.350472	0.350789	0.35059
	0.350564	0.350536	0.35041	0.350486	0.350478	0.35047	0.3508	0.350496
Alle - Doppelt	0.407473	0.407557	0.407406	0.407483	0.407467	0.407485	0.407862	0.407449
	0.407486	0.40758	0.407388	**0.407448**	0.407469	0.40747	0.407826	**0.407445**
	0.407463	0.407562	**0.407372**	0.407473	0.407447	0.407472	0.407787	0.407443
	0.407463	0.407538	0.407374	0.40749	0.407454	0.407469	0.407772	0.407525
	0.407465	0.407533	0.407373	0.407471	**0.407443**	0.407468	0.407744	0.407516
	0.407479	0.407551	0.407382	0.407476	0.407451	0.407468	0.407762	0.407459

Tabelle 5: Matching Ergebnisse für die Handgeste *Greifen* mit einem *Distance Threshold* von 0.002.

Konfiguration	Daumen	Faust	Greifen	**HandHinten**	HandVorne	Finger Zus.	OK	Victory
2nnk - Einzel	0.168463	0.16819	0.168209	0.167946	0.168119	0.168176	0.16857	0.16831
4nnk - Einzel	0.18207	0.181774	0.181788	0.181575	0.181708	0.181758	0.182098	0.181896
8nnk - Einzel	0.205134	0.204946	0.204953	0.204781	0.20486	0.204894	0.205206	0.205215
Alle - Einzel	0.238665	0.2385	0.238486	0.238339	0.2384	0.238466	0.238815	0.238511
	0.238671	0.238503	0.238447	**0.23831**	**0.238382**	0.238419	0.238731	0.238495
	0.238674	0.238488	0.238449	0.238355	0.238374	0.238408	0.238682	0.238714
2nnk - Doppelt	0.17435	0.173918	0.174003	0.173828	**0.173754**	0.173842	0.174361	0.173962
	0.174296	0.173896	0.174002	**0.173719**	0.17376	0.173832	0.174324	0.173965
4nnk - Doppelt	0.180942	0.180658	0.180653	0.180435	0.180577	0.180631	0.180938	0.180776
	0.180945	0.180623	0.180668	**0.180426**	0.18057	0.180603	0.180959	0.180695
8nnk - Doppelt	0.170305	**0.170035**	0.170089	**0.169871**	0.169967	0.170025	0.170291	0.17036
	0.170357	0.170054	0.170091	0.169878	0.169968	0.170007	0.17029	0.17013
Alle - Doppelt	0.159517	0.159244	0.15932	**0.159024**	0.159181	0.159262	0.159633	0.159372
	0.159458	0.159223	0.159334	0.159136	0.159184	0.159214	0.159584	0.159371
	0.159508	0.159219	0.159271	0.159037	0.159155	0.159208	0.159495	0.159366
	0.159482	0.15919	0.159291	0.159038	0.159154	0.159188	0.159512	0.159256
	0.159557	0.159196	0.159262	0.159037	0.159132	0.1592	0.159466	0.159534
	0.159599	0.159213	0.159259	0.159049	0.159135	0.159181	0.15948	0.159364

Tabelle 6: Matching Ergebnisse für die Handgeste *HandHinten* mit einem *Distance Threshold* von 0.002.

Konfiguration	Daumen	Faust	Greifen	HandHinten	**HandVorne**	Finger Zus.	OK	Victory
2nnk - Einzel	0.202348	0.202125	0.202199	0.202035	0.201903	0.201955	0.202407	0.202166
4nnk - Einzel	0.186434	0.186081	0.186272	0.186138	0.185967	0.186002	0.186343	0.186264
8nnk - Einzel	0.183216	0.182932	0.183045	0.182979	0.182799	0.182874	0.183173	0.183141
Alle - Einzel	0.182162	0.181932	0.182053	0.181881	0.18173	0.181805	0.182145	0.182
	0.182111	0.181852	0.182	0.181878	0.181729	**0.181779**	0.182102	0.182006
	0.182138	0.181828	0.18197	0.181896	**0.181722**	0.181786	0.182036	0.182075
2nnk - Doppelt	0.187433	0.187184	0.187327	0.187159	0.186988	**0.187069**	0.187445	0.187267
	0.187376	0.187142	0.187321	0.187125	**0.186978**	0.187074	0.187378	0.187267
4nnk - Doppelt	0.197367	0.19715	0.197256	0.197156	0.197011	0.197058	0.197388	0.197264
	0.197377	0.197146	0.197256	0.197159	**0.19701**	**0.197057**	0.197371	0.197182
8nnk - Doppelt	0.217637	0.217463	0.21753	0.217468	0.217325	0.217392	0.217654	0.217634
	0.217689	0.217489	0.217546	0.217472	0.217325	**0.21738**	0.217665	0.217519
Alle - Doppelt	0.210729	0.210554	0.21064	0.210509	0.210362	0.210443	0.210826	0.210595
	0.210686	0.210532	0.21063	0.210469	0.210358	0.210444	0.210763	0.210592
	0.210694	0.210498	0.210582	0.210499	0.210366	0.21042	0.210715	0.210584
	0.210711	0.210498	0.210586	0.2105	0.210366	**0.210416**	0.210699	0.21052
	0.210681	0.210482	0.210567	0.210518	0.210361	0.210426	0.210653	0.210681
	0.210734	0.210504	0.210583	0.210525	**0.210355**	0.210417	0.210661	0.210585

Tabelle 7: Matching Ergebnisse für die Handgeste *HandVorne* mit einem *Distance Threshold* von 0.002.

Konfiguration	Daumen	Faust	Greifen	HandHinten	HandVorne	**Finger Zus.**	OK	Victory
2nnk - Einzel	0.308832	0.30882	0.308777	0.308767	0.308685	0.30868	0.308984	0.308757
4nnk - Einzel	0.216189	0.216063	0.216127	0.216011	0.215949	0.215933	0.216276	0.216118
8nnk - Einzel	0.199399	0.199165	0.199294	0.199193	0.199078	0.199069	0.199361	0.199368
Alle - Einzel	0.284319	0.284309	0.284303	0.284229	**0.284139**	0.284146	0.284482	0.284262
	0.284305	0.284285	0.284265	0.28421	0.284147	0.284137	0.284449	0.284263
	0.284279	0.284256	0.284262	0.28423	0.284142	**0.284135**	0.284387	0.284302
2nnk - Doppelt	0.319914	0.319922	0.31985	0.319871	0.319786	**0.319779**	0.320113	0.319836
	0.319909	0.319924	0.319849	0.319867	**0.319785**	0.31978	0.320073	**0.319833**
4nnk - Doppelt	0.217453	0.217362	0.217383	0.2173	0.217223	0.217202	0.217609	0.217407
	0.21746	0.217355	0.217384	0.217307	**0.217221**	**0.2172**	0.217576	0.217359
8nnk - Doppelt	0.278169	0.278121	0.278131	0.27812	0.278022	0.278019	0.278255	0.278167
	0.278231	0.278139	0.278133	0.278123	**0.278019**	**0.278014**	0.278255	0.278132
Alle - Doppelt	0.234769	0.234697	0.23473	0.234614	0.234514	0.234533	0.234963	0.234688
	0.234738	0.234693	0.234731	0.234636	**0.234513**	0.234531	0.234909	0.234679
	0.23474	0.234665	0.23468	0.234599	0.234522	0.234516	0.234874	0.234676
	0.234753	0.234656	0.234679	0.2346	0.234521	0.234509	0.234843	0.234641
	0.234718	0.234637	0.234669	0.234622	0.234513	0.234518	0.234816	0.234736
	0.23478	0.234657	0.234676	0.234629	0.234513	**0.234507**	0.234818	0.234709

Tabelle 8: Matching Ergebnisse für die Handgeste *HandVorneFingerZusammen* mit einem *Distance Threshold* von 0.002.

Konfiguration	Daumen	Faust	Greifen	HandHinten	HandVorne	Finger Zus.	**OK**	Victory
2nnk - Einzel	0.277904	0.277724	0.277859	0.277655	0.277718	0.277692	0.277477	0.277674
4nnk - Einzel	0.319314	0.31928	0.319326	0.319211	0.319271	0.319211	0.31908	0.319261
8nnk - Einzel	0.304357	0.304221	0.304362	0.304235	0.304241	0.30423	0.304037	0.304444
Alle - Einzel	0.287175	0.287082	0.287158	0.287011	0.287027	0.287006	**0.286844**	0.287024
	0.287198	0.287043	0.287145	0.28702	0.28704	0.287014	0.286857	0.287046
	0.287149	0.287044	0.287149	0.287039	0.287037	0.287014	0.286865	0.287175
2nnk - Doppelt	0.319136	0.31905	0.319144	0.318978	0.319011	0.318994	**0.318794**	0.319032
	0.319119	0.319027	0.319186	0.31912	0.31903	0.319011	0.318804	0.319009
4nnk - Doppelt	0.324061	0.323939	0.324031	0.323886	0.323935	0.323912	**0.323708**	0.32391
	0.324023	0.32395	0.324068	0.323877	0.323902	0.323916	0.323712	0.323829
8nnk - Doppelt	0.368992	0.368929	0.369051	0.368925	0.368928	0.368912	0.368753	0.36915
	0.36898	0.36892	0.369027	0.368952	0.368915	0.368915	**0.368747**	0.368926
Alle - Doppelt	0.389372	0.389323	0.389412	0.389325	0.389325	0.389311	**0.389197**	0.389315
	0.389362	0.389314	0.389444	0.389399	0.389329	0.389334	0.389207	0.389311
	0.389382	0.389323	0.389414	0.389318	0.389335	0.38932	0.389219	0.389328
	0.389355	0.389323	0.389432	0.389313	0.389327	0.389324	0.389212	0.389294
	0.389368	0.38932	0.389441	0.389335	0.389329	0.389325	0.389221	0.389385
	0.389367	0.389314	0.389416	0.389358	0.389338	0.389323	0.389215	0.389381

Tabelle 9: Matching Ergebnisse für die Handgeste *OK* mit einem *Distance Threshold* von 0.002.

Konfiguration	Daumen	Faust	Greifen	HandHinten	HandVorne	Finger Zus.	OK	**Victory**
2nnk - Einzel	0.313407	0.313297	0.31332	0.313179	0.313298	0.313298	0.31354	0.313108
4nnk - Einzel	0.342229	0.342119	0.342101	0.342067	0.342127	0.342105	0.342358	0.341947
8nnk - Einzel	0.419719	0.419575	0.419539	0.419538	0.419502	0.419473	0.419913	0.419369
Alle - Einzel	0.445333	0.445184	0.445161	0.445156	0.445202	0.445132	0.445525	0.445253
	0.445306	0.44519	0.44513	0.445126	0.445147	0.445121	0.445493	0.445236
	0.445276	0.445174	0.445119	0.445142	0.445122	0.445107	0.445456	**0.445028**
2nnk - Doppelt	0.337673	0.337587	0.337575	0.337471	0.33758	0.337557	0.337842	**0.337346**
	0.337689	0.337579	0.337529	0.337495	0.337573	0.337554	0.337818	0.337373
4nnk - Doppelt	0.31028	0.310089	0.310081	0.310035	0.310107	0.310094	0.310335	**0.30994**
	0.310258	0.310077	0.310073	**0.310032**	0.310102	0.310081	0.310357	0.310114
8nnk - Doppelt	0.42282	0.422643	0.422596	0.422611	0.422582	0.422558	0.422929	**0.422478**
	0.422824	0.42264	0.422608	0.422616	0.422614	0.422589	0.422947	0.422741
Alle - Doppelt	0.41719	0.416953	0.41691	0.416908	0.41696	0.416864	0.417353	0.417005
	0.417207	0.417	0.416914	0.416911	0.416916	0.41687	0.417315	0.417005
	0.417149	0.416952	0.416879	0.416865	0.416879	0.416847	0.417265	0.416983
	0.417154	0.416944	0.416893	0.416867	0.416896	0.416857	0.417252	0.417027
	0.417091	0.416931	0.416872	0.416897	0.416847	0.41683	0.41723	**0.416764**
	0.417095	0.416925	0.416881	0.41691	0.416888	0.416854	0.417248	0.417031

Tabelle 10: Matching Ergebnisse für die Handgeste *Victory* mit einem *Distance Threshold* von 0.002.

Konfiguration	**Daumen**	Faust	Greifen	HandHinten	HandVorne	Finger Zus.	OK	Victory
2nnk - Einzel	0.322962	0.323147	0.323293	0.323299	0.323348	0.323325	0.323205	0.323048
4nnk - Einzel	0.325004	0.325233	0.325354	0.325387	0.325393	0.325361	0.325365	0.325188
8nnk - Einzel	0.302203	0.302444	0.302501	0.302696	0.302686	0.302689	0.302491	0.302439
Alle - Einzel	0.409952	0.410058	0.410147	0.410209	0.410211	0.410203	0.410191	0.41007
	0.409915	0.410059	0.410145	0.410203	0.410202	0.410173	0.410222	0.410079
	0.409887	0.410079	0.410154	0.4102	0.41021	0.410193	0.410157	0.410055
2nnk - Doppelt	0.306928	0.307043	0.30712	0.307207	0.307188	0.307185	0.307099	0.307
	0.306906	0.307055	0.307105	0.307188	0.307199	0.30721	0.307103	0.307039
4nnk - Doppelt	**0.346729**	0.346918	0.34698	0.34711	0.347114	0.347086	0.347041	0.346928
	0.346743	0.346937	0.346986	0.347106	0.347118	0.347093	0.347074	0.346929
8nnk - Doppelt	**0.331758**	0.331953	0.332051	0.33213	0.332104	0.332109	0.331993	0.331919
	0.331812	0.331934	0.332046	0.33212	0.332099	0.33211	0.331995	0.331896
Alle - Doppelt	0.37826	0.378339	0.378408	0.378557	0.378536	0.378531	0.378417	0.378312
	0.378215	0.378359	0.378406	0.378512	0.37856	0.378564	0.378421	0.378341
	0.378198	0.378322	0.378408	0.378541	0.378527	0.3785	0.378452	0.378339
	0.378208	0.378334	0.378413	0.37852	0.378536	0.378505	0.378488	0.378347
	0.378164	0.378334	0.378399	0.378527	0.378536	0.378524	0.378387	0.378302
	0.378228	0.378324	0.378443	0.378528	0.378516	0.378519	0.378387	0.378295

Tabelle 11: Matching Ergebnisse für die Handgeste *Daumen* mit einem *Distance Threshold* von 0.005.

Konfiguration	Daumen	Faust	Greifen	HandHinten	**HandVorne**	Finger Zus.	OK	Victory
2nnk - Einzel	0.152139	0.151736	0.15194	0.151597	0.151521	0.151578	0.151824	0.1519
4nnk - Einzel	0.152347	0.151796	0.152011	0.15165	0.151585	0.151637	0.151819	0.15204
8nnk - Einzel	0.213661	0.213389	0.213591	0.213327	0.213265	0.213313	0.213442	0.21352
Alle - Einzel	0.210251	0.20999	0.210137	0.209887	0.209818	0.20987	0.210064	0.210027
	0.210276	0.20998	0.210125	**0.209883**	0.209816	0.20984	0.210055	0.210006
	0.210248	0.209957	0.210158	0.209899	**0.209811**	**0.209846**	0.209993	0.210007
2nnk - Doppelt	0.170802	0.170462	0.170648	**0.170337**	0.170296	0.170338	0.170526	0.17063
	0.170811	0.170443	0.170636	0.170342	**0.170271**	**0.170326**	0.170482	0.170653
4nnk - Doppelt	0.162954	0.162415	0.162649	**0.162271**	**0.162208**	**0.162265**	0.162441	0.162628
	0.163005	0.162381	0.162677	0.162285	0.162209	0.162267	0.162477	0.162672
8nnk - Doppelt	0.19431	0.193962	0.194192	0.193914	**0.193845**	**0.193905**	0.19402	0.194124
	0.194304	0.193972	0.194088	0.193914	0.193853	0.193906	0.194053	0.194093
Alle - Doppelt	0.216344	0.216154	0.216266	0.216066	0.21606	0.216091	0.216227	0.216248
	0.216368	0.216175	0.216258	0.216069	0.216051	0.216086	0.216192	0.216246
	0.216416	0.216149	0.216265	**0.216063**	0.216056	0.21608	0.216205	0.216239
	0.216437	0.21615	0.216263	0.21607	0.216052	**0.216078**	0.216227	0.216252
	0.216365	0.216143	0.216289	0.216085	**0.216045**	0.216083	0.216161	0.216228
	0.216364	0.216144	0.216199	0.21608	0.216052	0.216083	0.216196	0.216215

Tabelle 12: Matching Ergebnisse für die Handgeste *HandVorne* mit einem *Distance Threshold* von 0.005.

Konfiguration	Daumen	Faust	Greifen	HandHinten	HandVorne	Finger Zus.	**OK**	Victory
2nnk - Einzel	0.223405	0.2234	0.223495	0.223406	0.223372	0.223412	0.223181	0.223383
4nnk - Einzel	0.268528	0.268629	0.268694	0.268621	0.268575	0.268599	0.268407	0.268645
8nnk - Einzel	0.290719	0.29073	0.290863	0.290825	0.29067	0.290713	0.290569	0.290695
Alle - Einzel	0.27235	0.272392	0.272519	0.27246	0.272413	0.272375	0.272215	0.272326
	0.272372	0.272391	0.272489	0.272456	0.272332	0.272324	0.272206	0.272344
	0.272328	0.272351	0.272477	0.272474	0.272329	0.272355	**0.272204**	0.272315
2nnk - Doppelt	0.278275	0.278268	0.278379	0.278296	0.278236	0.278263	**0.278059**	0.278219
	0.278267	0.278218	0.278402	0.278293	0.278212	0.278271	0.278069	0.278241
4nnk - Doppelt	0.286383	0.286404	0.286524	0.286444	0.286341	0.286367	0.286212	0.28639
	0.286371	0.286397	0.286557	0.286464	0.286334	0.28637	**0.286207**	0.286393
8nnk - Doppelt	0.260168	0.260189	0.260348	0.260282	0.260125	0.260204	**0.259998**	0.260175
	0.260169	0.26022	0.260303	0.260276	0.260131	0.2602	0.260002	0.260119
Alle - Doppelt	0.250218	0.250243	0.250335	0.250214	0.250149	0.250218	0.249958	0.250173
	0.250212	0.250168	0.250357	0.250213	0.25014	0.250227	0.249966	0.250212
	0.25016	0.250231	0.250316	0.25021	0.25013	0.250162	0.249954	0.250206
	0.25017	0.25022	0.250359	0.250231	0.250115	0.250168	0.249951	0.250216
	0.250125	0.25018	0.25032	0.250251	0.250098	0.25018	0.249937	0.25016
	0.250136	0.250212	0.250264	0.250246	0.250092	0.250176	**0.249934**	0.2501

Tabelle 13: Matching Ergebnisse für die Handgeste *OK* mit einem *Distance Threshold* von 0.005.

Literatur

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006.

[2] M. Hansard, S. Lee, O. Choi, and R. Horaud. *Time-of-Flight Cameras: Principles, Methods and Applications*. Springer, Dez 2012.

[3] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.

[4] P. Kumar M. Connor. Fast construction of k-nearest neighbor graphs for point clouds. Discussion paper, Florida State University ,Department of Computer Science, September 2009.

[5] Point cloud library. `http://pointclouds.org/`. Stand: 18.01.2015.

[6] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edingburgh and Dublin Philosophical Magazine and Journal of Science, Sixth Series*, 2:559–572, 1901.

[7] S. Preuß. Von Punktwolken zu Dreiecksnetzen. Diplomarbeit, Universität Karlsruhe, Jan 2002.

[8] A. Steger. *Diskrete Strukturen: Kombinatorik, Graphentheorie, Algebra*. Springer-Verlag, 2007.

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit.

Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

_____ _____

Datum Unterschrift